

PTO/SB/21 (07-06)

Approved for use through 09/30/2006. OMB 0651-0031

U.S. Patent and Trademark Office; U.S. DEPARTMENT OF COMMERCE

Under the Paperwork Reduction Act of 1995, no persons are required to respond to a collection of information unless it displays a valid OMB control number.

**TRANSMITTAL
FORM**

(to be used for all correspondence after initial filing)

Total Number of Pages in This Submission

979

Application Number

09/912,636

Filing Date

07/24/2001

First Named Inventor

Elliot Schwartz

Art Unit

2142

Examiner Name

Thong H. Vu


Attorney Docket Number

005168.P002

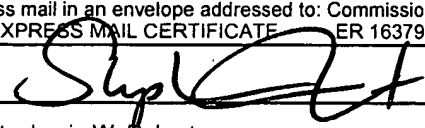
ENCLOSURES (Check all that apply)

<input checked="" type="checkbox"/> Fee Transmittal Form	<input type="checkbox"/> Drawing(s)	<input type="checkbox"/> After Allowance Communication to TC
<input checked="" type="checkbox"/> Fee Attached	<input type="checkbox"/> Licensing-related Papers	<input type="checkbox"/> Appeal Communication to Board of Appeals and Interferences
<input checked="" type="checkbox"/> Amendment/Reply	<input type="checkbox"/> Petition	<input checked="" type="checkbox"/> Appeal Communication to TC (Appeal Notice, Brief, Reply Brief)
<input type="checkbox"/> After Final	<input type="checkbox"/> Petition to Convert to a Provisional Application	<input type="checkbox"/> Proprietary Information
<input type="checkbox"/> Affidavits/declaration(s)	<input type="checkbox"/> Power of Attorney, Revocation	<input type="checkbox"/> Status Letter
<input type="checkbox"/> Extension of Time Request	<input type="checkbox"/> Change of Correspondence Address	<input checked="" type="checkbox"/> Other Enclosure(s) (please identify below):
<input type="checkbox"/> Express Abandonment Request	<input type="checkbox"/> Terminal Disclaimer	Please see remarks below.
<input type="checkbox"/> Information Disclosure Statement	<input type="checkbox"/> Request for Refund	
<input type="checkbox"/> Certified Copy of Priority Document(s)	<input type="checkbox"/> CD, Number of CD(s) _____	
<input type="checkbox"/> Reply to Missing Parts/Incomplete Application	<input type="checkbox"/> Landscape Table on CD	
<input type="checkbox"/> Reply to Missing Parts under 37 CFR 1.52 or 1.53		
Remarks 1. Transmittal form SB/21. (1 page) 2. Fee Transmittal SB/17. (1 page) 3. 2402 Payment for Filing a Brief in Support of An Appeal via PTO-2038. (1 page) 4. Appeal Brief (974 pages) 5. Return Receipt PostCard. (1 page) 6. Express Certificate of Mailing (1 page)		

SIGNATURE OF APPLICANT, ATTORNEY, OR AGENT

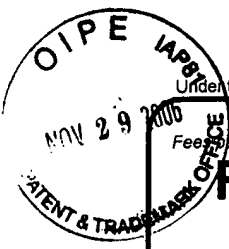
Firm Name	Heimlich Law		
Signature	 Digitally signed by Alan Heimlich DN: cn = Alan Heimlich, o = US, o = Heimlich Law		
Printed name	Alan Heimlich		
Date	11/29/2006	Reg. No.	48808

CERTIFICATE OF TRANSMISSION/MAILING

I hereby certify that this correspondence is being facsimile transmitted to the USPTO or deposited with the United States Postal Service with sufficient postage as first class mail in an envelope addressed to: Commissioner for Patents, P.O. Box 1450, Alexandria, VA 22313-1450 on the date shown below: EXPRESS MAIL CERTIFICATE ER 163796141 US			
Signature			
Typed or printed name	Stephanie W. Roberts	Date	11/29/2006

This collection of information is required by 37 CFR 1.5. The information is required to obtain or retain a benefit by the public which is to file (and by the USPTO to process) an application. Confidentiality is governed by 35 U.S.C. 122 and 37 CFR 1.11 and 1.14. This collection is estimated to 2 hours to complete, including gathering, preparing, and submitting the completed application form to the USPTO. Time will vary depending upon the individual case. Any comments on the amount of time you require to complete this form and/or suggestions for reducing this burden, should be sent to the Chief Information Officer, U.S. Patent and Trademark Office, U.S. Department of Commerce, P.O. Box 1450, Alexandria, VA 22313-1450. DO NOT SEND FEES OR COMPLETED FORMS TO THIS ADDRESS. SEND TO: Commissioner for Patents, P.O. Box 1450, Alexandria, VA 22313-1450.

If you need assistance in completing the form, call 1-800-PTO-9199 and select option 2.



Effective on 12/08/2004.
Fees pursuant to the Consolidated Appropriations Act, 2005 (H.R. 4818).

FEE TRANSMITTAL

For FY 2006

☒ Applicant claims small entity status. See 37 CFR 1.27

TOTAL AMOUNT OF PAYMENT (\$) 250.00

Complete if Known

Application Number	09/912,636
Filing Date	07/24/2001
First Named Inventor	Elliot Schwartz
Examiner Name	Thong H. Vu
Art Unit	2142
Attorney Docket No.	005168.P002

METHOD OF PAYMENT (check all that apply)

☐ Check ☒ Credit Card ☐ Money Order ☐ None ☐ Other (please identify): _____

☐ Deposit Account Deposit Account Number: _____ Deposit Account Name: _____

For the above-identified deposit account, the Director is hereby authorized to: (check all that apply)

☒ Charge fee(s) indicated below ☐ Charge fee(s) indicated below, except for the filing fee
☒ Charge any additional fee(s) or underpayments of fee(s) under 37 CFR 1.16 and 1.17 ☒ Credit any overpayments

WARNING: Information on this form may become public. Credit card information should not be included on this form. Provide credit card information and authorization on PTO-2038.

FEE CALCULATION

1. BASIC FILING, SEARCH, AND EXAMINATION FEES

Application Type	FILING FEES		SEARCH FEES		EXAMINATION FEES		Fees Paid (\$)
	Fee (\$)	Small Entity Fee (\$)	Fee (\$)	Small Entity Fee (\$)	Fee (\$)	Small Entity Fee (\$)	
Utility	300	150	500	250	200	100	_____
Design	200	100	100	50	130	65	_____
Plant	200	100	300	150	160	80	_____
Reissue	300	150	500	250	600	300	_____
Provisional	200	100	0	0	0	0	_____

2. EXCESS CLAIM FEES

Fee Description	Fee (\$)	Small Entity Fee (\$)
Each claim over 20 (including Reissues)	50	25
Each independent claim over 3 (including Reissues)	200	100
Multiple dependent claims	360	180
Total Claims	Extra Claims	Fee (\$)
_____ - 20 or HP = _____	x _____	= _____
HP = highest number of total claims paid for, if greater than 20.		
Indep. Claims	Extra Claims	Fee (\$)
_____ - 3 or HP = _____	x _____	= _____
HP = highest number of independent claims paid for, if greater than 3.		

3. APPLICATION SIZE FEE

If the specification and drawings exceed 100 sheets of paper (excluding electronically filed sequence or computer listings under 37 CFR 1.52(e)), the application size fee due is \$250 (\$125 for small entity) for each additional 50 sheets or fraction thereof. See 35 U.S.C. 41(a)(1)(G) and 37 CFR 1.16(s).

Total Sheets _____ Extra Sheets _____ Number of each additional 50 or fraction thereof _____ Fee (\$)

_____ - 100 = _____ / 50 = _____ (round up to a whole number) x _____ = _____ Fee Paid (\$)

4. OTHER FEE(S)

Non-English Specification, \$130 fee (no small entity discount)

Other (e.g., late filing surcharge): 2402 Payment for Filing a Brief in Support of An Appeal

Fees Paid (\$)

\$250.00

SUBMITTED BY

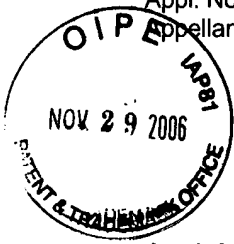
Signature		Digitally signed by Alan Heimlich DN: CN = Alan Heimlich, C = US, O = Heimlich Law	Registration No. (Attorney/Agent) 48808	Telephone 408 253 3860
Name (Print/Type)	Alan Heimlich			Date 29 November 2006

This collection of information is required by 37 CFR 1.136. The information is required to obtain or retain a benefit by the public which is to file (and by the USPTO to process) an application. Confidentiality is governed by 35 U.S.C. 122 and 37 CFR 1.14. This collection is estimated to take 30 minutes to complete, including gathering, preparing, and submitting the completed application form to the USPTO. Time will vary depending upon the individual case. Any comments on the amount of time you require to complete this form and/or suggestions for reducing this burden, should be sent to the Chief Information Officer, U.S. Patent and Trademark Office, U.S. Department of Commerce, P.O. Box 1450, Alexandria, VA 22313-1450. DO NOT SEND FEES OR COMPLETED FORMS TO THIS ADDRESS. SEND TO: Commissioner for Patents, P.O. Box 1450, Alexandria, VA 22313-1450.

If you need assistance in completing the form, call 1-800-PTO-9199 and select option 2.

Appl. No. 09/912,636

Appellant's Brief



IN THE UNITED STATES PATENT AND TRADEMARK OFFICE

BEFORE THE BOARD OF PATENT APPEALS AND INTERFERENCES

Appl. No. : 09/912,636
Applicant : Elliot Schwartz
Filed : 07/24/2001
TC/A.U. : 2142
Examiner : Thong H. Vu
Confirmation No. : 6340

Docket No. : 005168.P002
Customer No. : 40418

Title : Network architecture

COMMUNICATED VIA EXPRESS MAIL PO TO ADDRESSEE ER 163796141 US

APPELLANT'S APPEAL BRIEF Under 37 C.F.R. § 41.37

Dear Sir:

Applicant (Appellant) hereby submits this Appeal Brief pursuant to 37 C.F.R. § 41.37 in connection with the above-referenced application and respectfully requests consideration by the Board of Patent Appeals and Interferences for allowance from a fourth rejection decision by the Examiner. The Examiner's fourth rejection decision ("Office Action" or "Office") was mailed on March 23, 2006 and rejected all claims (1-27). Applicant also submits herewith a credit card authorization or payment in the amount of \$250 as the fee for filing an Appeal Brief required by 37 C.F.R. § 41.20(b)(2). Applicant submitted a Notice of Appeal that was received on September 29, 2006.

11/30/2006 EAREGAY1 00000076 09912636

01 FC:2402

250.00 OP

TABLE OF CONTENTS

	<u>Page</u>
I. REAL PARTY IN INTEREST	4
II. RELATED APPEALS AND INTERFERENCES.....	4
III. STATUS OF CLAIMS	4
IV. STATUS OF AMENDMENTS.....	4
V. SUMMARY OF CLAIMED SUBJECT MATTER	5-11
VI. GROUNDS OF REJECTION TO BE REVIEWED ON APPEAL	12
VII. ARGUMENT.....	13-76
PREFACE.....	13
Historical prosecution history	14-15
INTRODUCTION	16
Claims 1-27 Rejection under 35 U.S.C. § 101	17
Claim 1 Rejection under 35 U.S.C. § 101	17-20
Claims 2-9 Rejection under 35 U.S.C. § 101	21
Claim 10 Rejection under 35 U.S.C. § 101	21-22
Claims 11-18 Rejection under 35 U.S.C. § 101	22
Claim 19 Rejection under 35 U.S.C. § 101	22-23
Claims 20-27 Rejection under 35 U.S.C. § 101	23
Claim 1 Rejection under 35 U.S.C. § 102(e) – Bavadekar.....	24-28
Claim 2 Rejection under 35 U.S.C. § 102(e) – Bavadekar.....	29-30
Claims 10-11, 19-20 Rejection under 35 U.S.C. § 102(e) - Bavadekar	31
Claim 3 Rejection under 35 U.S.C. § 102(e) – Bavadekar - Pujare	31-34
Claim 4 Rejection under 35 U.S.C. § 102(e) – Bavadekar - Pujare	34-36
Claim 5 Rejection under 35 U.S.C. § 102(e) – Bavadekar - Pujare	36-38
Claim 6 Rejection under 35 U.S.C. § 102(e) – Bavadekar - Pujare	38-40
Claim 7 Rejection under 35 U.S.C. § 102(e) – Bavadekar - Pujare	40-42
Claim 8 Rejection under 35 U.S.C. § 102(e) – Bavadekar - Pujare	42-44
Claim 9 Rejection under 35 U.S.C. § 102(e) – Bavadekar - Pujare	44-46
Claims 12-18, 21-27 Rejection under 35 U.S.C. § 102(e) - Bavadekar	47
Conclusion.....	48

VIII.	CLAIMS APPENDIX	49-53
IX.	EVIDENCE APPENDIX.....	54-973
	(A) Evidence for Claims 1-27 – Relied Upon	54-88
	(B) Evidence for Claims 4-9 – Relied Upon	89-175
	(C) Evidence for Claims 1-27 – Entered by Examiner.....	176-807
	(D) Art submitted by Applicant - Entered	808-973
X.	RELATED PROCEEDINGS	974

I. REAL PARTY IN INTEREST

The real party in interest of Appellant is Digi International, Incorporated.

II. RELATED APPEALS AND INTERFERENCES

Appellant is unaware of any related appeals or interferences.

1) There are no prior or pending interferences.

2) There are no pending appeals before the USPTO.

3) There is no judicial proceeding related to Appellant's instant application or related applications.

III. STATUS OF THE CLAIMS

Claims 1-27 are pending in the application. No claims have been allowed.

All claims are on appeal.

All claims have been rejected by the Examiner as follows.

Claims 1-27 are rejected under 35 U.S.C. § 101 because the claimed invention is directed to non-statutory subject matter. Claims 1-27 are rejected under 35 U.S.C. § 102(e) as being anticipated by Bavadekar (U.S. Patent Application No. 2003/0009571 A1).

IV. STATUS OF AMENDMENTS

No amendments have been filed after receipt of the fourth rejection.

//

V. SUMMARY OF THE CLAIMED SUBJECT MATTER

Appellant has indicated below representative Figures/Specifications – not all are included so as to keep the claims brief. Appellant has indicated in larger font the three independent claims (1, 10, and 19).

1. A computer network architecture (Figure 2. Specification: page 8, lines 17-18; page 30, line 1.) **comprising:**

a first layer including a transmission control protocol connection;

(Figure 2 at 222 (Transport Layer). Specification: page 8, line 22; page 30, line 2.)

a second layer including a hyper text transfer protocol connection

(Figure 2 at 232 (HTTP). Specification: page 9, lines 3-5; page 30, line 3.) **built upon the first layer;**

a first tunneling layer including a first tunneling protocol built upon the second layer to tunnel a message through the hyper text transfer protocol connection; (Figure 2 at 240 (MDH). Specification: page 9, lines 3-5.) **and**

a multiplexing layer to multiplex a plurality of messages for transmission through the first tunneling layer. (Figure 2 at 270 (Multiplexing Layer). Specification: page 9, lines 12-14.)

2. The computer network architecture of claim 1, wherein the first tunneling protocol opens the hyper text transfer protocol connection between a server and a client. (Figure 4, at 410, 420. Specification: page 10, lines 16-21.)

**3. The computer network architecture of claim 1, further comprising:
a second tunneling layer including a second tunneling protocol** (Figure 2 at 226. Specification: page 9, lines 6-7.) **built upon the first layer to tunnel a message through the transmission control protocol connection.** (Figure 2 at 222. Specification: page 8, line 22.)

4. The computer network architecture of claim 3, wherein the second tunneling protocol (Figure 2 at 226. Specification: page 9, lines 6-7) **is used to open the transmission control protocol connection between the server and the client.** (Figure 4 at 422. Specification: page 10, lines 7-15.)

5. The computer network architecture of claim 4, wherein the first tunneling protocol (Figure 11 at 1140 (MDH protocol).) **opens the hyper text transfer protocol connection if the second tunneling protocol** (Figure 11 at 1110 (MT protocol).) **is not successful in opening the transmission control protocol connection** (Figure 11 at 1120.). (Figure 11. Specification: page 22, line 19 to page 23, line 8.)

6. The computer network of claim 1, wherein the messages include binary format messages. (Specification: page 7, line 18.)

7. The computer network architecture of claim 1, wherein the plurality of messages includes a plurality of operational messages (Specification: page 17, line 11.) and a plurality of administrative messages. (Specification: page 18, lines 2, 10-11.)

8. The computer network architecture of claim 7, wherein the operational messages include operational data. (Specification: page 18, line 16.)

9. The computer network architecture of claim 7, wherein the administrative messages can be selected from the group consisting of debug messages, firmware update messages and parameter configuration messages. (Specification: page 26, lines 1-3.)

10. A method for a computer network architecture comprising:
building a hyper text transfer protocol connection upon (Figure 2 at 232 (HTTP). Specification: page 9, lines 3-5; page 30, line 3.) a transmission control protocol connection; (Figure 2 at 222 (Transport Layer). Specification: page 8, line 22; page 30, line 2.)

tunneling a message through the hyper text transfer protocol connection by using a first tunneling protocol layer including a first tunneling protocol; (Figure 2 at 240 (MDH). Specification: page 9, lines 3-5.) and

multiplexing a plurality of messages for transmission through the hyper text transfer protocol connection by using a multiplexing layer.

(Figure 2 at 270 (Multiplexing Layer). Specification: page 9, lines 12-14.)

11. The method of claim 10, wherein opening the hyper text transfer protocol connection between a server and a client by using the first tunneling layer. (Figure 5 at 502 (MDH Connection Request). Specification: page 11, lines 2-3.)

12. The method of claim 10, further comprising:

tunneling a message through the transmission control protocol connection (Figure 2 at 222. Specification: page 8, line 22.) **by using a second tunneling protocol layer including a second tunneling protocol built upon** (Figure 2 at 226 (MT).

Specification: page 9, lines 6-7.) the transmission control protocol connection. (Figure 2 at 222. Specification: page 8, line 22.)

13. The method of claim 12, wherein opening the transmission control protocol connection between a server and a client by using the second tunneling protocol.

(Figure 11 at 1110. Specification: page 22, lines 20-21.)

14. The method of claim 13, wherein opening the hyper text transfer protocol connection by using the first tunneling protocol if the transmission control protocol connection is not successfully opened by using the second tunneling protocol.

(Figure 11 at 1120, 1140. Specification: page 23, lines 2-4.)

- 15. The method of claim 10, wherein the messages include binary format messages.** (Specification: page 7, line 18.)
- 16. The method of claim 10, wherein the plurality of messages include a plurality of operational messages** (Specification: page 17, line 11.) **and a plurality of administrative messages.** (Specification: page 18, lines 2, 10-11.)
- 17. The method of claim 16, wherein the operational messages include operational data.** (Specification: page 18, line 16.)
- 18. The method of claim 16, wherein the administrative messages can be selected from the group consisting of debug messages, firmware update messages and parameter configuration messages.** (Specification: page 27, lines 19-21.)
- 19. A computer readable medium** (Figure 10 at 1006, 1008, 1010. Specification: page 23, lines 15-16.) **having instructions which, when executed by a processing system** (Figure 10.) , **cause the system to perform a method comprising:**
- building a hyper text transfer protocol connection upon** (Figure 2 at 232 (HTTP). Specification: page 9, lines 3-5; page 30, line 3.) **a transmission control protocol connection;** (Figure 2 at 222 (Transport Layer). Specification: page 8, line 22; page 30, line 2.)

tunneling a message through the Hyper Text Transfer Protocol connection by using a first tunneling protocol layer including a first tunneling protocol; (Figure 2 at 240 (MDH). Specification: page 9, lines 3-5.) **and multiplexing a plurality of messages for transmission through the Hyper Text Transfer Protocol connection by using a multiplexing layer.** (Figure 2 at 270 (Multiplexing Layer). Specification: page 9, lines 12-14.)

20. The medium of claim 19, wherein opening the Hyper Text Transfer Protocol connection between a server and a client by using the first tunneling layer. (Figure 5 at 502 (MDH Connection Request). Specification: page 11, lines 2-3.)

21. The medium of claim 19, further comprising:

tunneling a message through the transmission control protocol connection (Figure 2 at 222. Specification: page 8, line 22.) **by using a second tunneling protocol layer including a second tunneling protocol built upon** (Figure 2 at 226 (MT). Specification: page 9, lines 6-7.) **the Transmission Control Protocol Connection.** (Figure 2 at 222. Specification: page 8, line 22.)

22. The medium of claim 21, wherein opening the Transmission Control Protocol connection between a server and a client by using the second tunneling protocol. (Figure 11 at 1110. Specification: page 22, lines 20-21.)

23. The medium of claim 22, wherein opening the Hyper Text Transfer Protocol connection by using the first tunneling protocol (Figure 11 at 1120 (NO), 1140.) if the Transmission Control Protocol connection is not successfully opened by using the second tunneling protocol. (Figure 11 at 1120 (NO), 1140. Specification: page 23, lines 2-4.)

24. The medium of claim 19, wherein the messages include binary format messages. (Specification: page 7, line 18.)

25. The medium of claim 19, wherein the plurality of messages include a plurality of operational messages (Specification: page 17, line 11.) and a plurality of administrative messages. (Specification: page 18, lines 2, 10-11.)

26. The medium of claim 25, wherein the operational messages include operational data. (Specification: page 18, line 16.)

27. The medium of claim 25, wherein the administrative messages can be selected from the group consisting of debug messages, firmware update messages and parameter configuration messages. (Specification: page 29, lines 15-17.)

//

VI. GROUNDS OF REJECTION TO BE REVIEWED ON APPEAL

The issues presented on appeal are whether Applicant's claims 1-27 for the reasons stated in the Status of the Claims are unpatentable.

First

Claims 1-27 stand rejected under 35 U.S.C. § 101 as being directed to non-statutory subject matter.

Second

Claims 1-27 stand rejected under 35 U.S.C. § 102(e) as being anticipated by Bavadekar (U.S. Patent Application No. 2003/0009571 A1.

//

VII. ARGUMENT

PREFACE

Board of Appeals,

It is this type of examination on the instant application that is driving us conscientious patent attorneys up the wall.

We have had 3 prior office actions where the Applicant has successfully overcome the Examiner's arguments without amending any claims only to be met with a new rejection based on "applicant's arguments moot in light of new art."

Now on the 4th rejection, the Examiner is raising for the first time a 101 rejection. What kind of examination is being done by this Examiner? A 101 should have been raised previously. A response by the examiner of "only now do I understand what applicant is trying to claim" is a lame excuse as Applicant has not amended anything.

Applicant respectfully requests that this Board please read the application and correspondence and move this case forward.

Extremely frustrated,

Alan Heimlich

Historical prosecution history

For the benefit of the board, the claims and the historical prosecution history of the instant application are illustrated on the following page. Claim structure is illustrated by indentation of the claims. Leftmost claims are independent. (I.e. claim 2 is dependent on claim 1, as is claim 3. Claim 5 is dependent on 4, which is dependent on 3, which is dependent on 1.)

Each claim is addressed below, however, Claim 1 is considered particularly relevant because the dependent claims and other independent claims and dependent claims are variations on claim 1.

Digi P002 Appeal historical claim tree.doc

CLAIM #	4 th Rejection 03-23-2006	3 rd OA Rejection 08-12-2005	2 nd OA Rejection 01-31-2005	1 st OA Rejection 10-06-2004
1	101 nonstatutory; 102(e) Bavadekar	103(a) Noy ivo Jacob	102(e)/103(a) Urien	103(a) Jacob ivo Balabine
2	101 nonstatutory; 102(e) Bavadekar	103(a) Noy ivo Jacob	102(e)/103(a) Urien	103(a) Jacob ivo Balabine
3	101 nonstatutory; 102(e) Bavadekar	103(a) Noy ivo Jacob fivo Pujare	102(e)/103(a) Urien	103(a) Jacob ivo Balabine
4	101 nonstatutory; 102(e) Bavadekar-Pujare?	103(a) Noy ivo Jacob fivo Pujare	102(e)/103(a) Urien	103(a) Jacob ivo Balabine
5	101 nonstatutory; 102(e) Bavadekar-Pujare?	103(a) Noy ivo Jacob fivo Pujare	102(e)/103(a) Urien	103(a) Jacob ivo Balabine
6	101 nonstatutory; 102(e) Bavadekar-Pujare?	103(a) Noy ivo Jacob fivo Pujare	102(e)/103(a) Urien	103(a) Jacob ivo Balabine
7	101 nonstatutory; 102(e) Bavadekar-Pujare?	103(a) Noy ivo Jacob fivo Pujare	102(e)/103(a) Urien	103(a) Jacob ivo Balabine
8	101 nonstatutory; 102(e) Bavadekar-Pujare?	103(a) Noy ivo Jacob fivo Pujare	102(e)/103(a) Urien	103(a) Jacob ivo Balabine
9	101 nonstatutory; 102(e) Bavadekar-Pujare?	103(a) Noy ivo Jacob fivo Pujare	102(e)/103(a) Urien	103(a) Jacob ivo Balabine
10	101 nonstatutory; 102(e) Bavadekar	103(a) Noy ivo Jacob	102(e)/103(a) Urien	103(a) Jacob ivo Balabine
11	101 nonstatutory; 102(e) Bavadekar	103(a) Noy ivo Jacob	102(e)/103(a) Urien	103(a) Jacob ivo Balabine
12	101 nonstatutory; 102(e) Bavadekar	103(a) Noy ivo Jacob fivo Pujare	102(e)/103(a) Urien	103(a) Jacob ivo Balabine
13	101 nonstatutory; 102(e) Bavadekar	103(a) Noy ivo Jacob fivo Pujare	102(e)/103(a) Urien	103(a) Jacob ivo Balabine
14	101 nonstatutory; 102(e) Bavadekar	103(a) Noy ivo Jacob fivo Pujare	102(e)/103(a) Urien	103(a) Jacob ivo Balabine
15	101 nonstatutory; 102(e) Bavadekar	103(a) Noy ivo Jacob fivo Pujare	102(e)/103(a) Urien	103(a) Jacob ivo Balabine
16	101 nonstatutory; 102(e) Bavadekar	103(a) Noy ivo Jacob fivo Pujare	102(e)/103(a) Urien	103(a) Jacob ivo Balabine
17	101 nonstatutory; 102(e) Bavadekar	103(a) Noy ivo Jacob fivo Pujare	102(e)/103(a) Urien	103(a) Jacob ivo Balabine
18	101 nonstatutory; 102(e) Bavadekar	103(a) Noy ivo Jacob fivo Pujare	102(e)/103(a) Urien	103(a) Jacob ivo Balabine
19	101 nonstatutory; 102(e) Bavadekar	103(a) Noy ivo Jacob	102(e)/103(a) Urien	103(a) Jacob ivo Balabine
20	101 nonstatutory; 102(e) Bavadekar	103(a) Noy ivo Jacob	102(e)/103(a) Urien	103(a) Jacob ivo Balabine
21	101 nonstatutory; 102(e) Bavadekar	103(a) Noy ivo Jacob fivo Pujare	102(e)/103(a) Urien	103(a) Jacob ivo Balabine
22	101 nonstatutory; 102(e) Bavadekar	103(a) Noy ivo Jacob fivo Pujare	102(e)/103(a) Urien	103(a) Jacob ivo Balabine
23	101 nonstatutory; 102(e) Bavadekar	103(a) Noy ivo Jacob fivo Pujare	102(e)/103(a) Urien	103(a) Jacob ivo Balabine
24	101 nonstatutory; 102(e) Bavadekar	103(a) Noy ivo Jacob fivo Pujare	102(e)/103(a) Urien	103(a) Jacob ivo Balabine
25	101 nonstatutory; 102(e) Bavadekar	103(a) Noy ivo Jacob fivo Pujare	102(e)/103(a) Urien	103(a) Jacob ivo Balabine
26	101 nonstatutory; 102(e) Bavadekar	103(a) Noy ivo Jacob fivo Pujare	102(e)/103(a) Urien	103(a) Jacob ivo Balabine
27	101 nonstatutory; 102(e) Bavadekar	103(a) Noy ivo Jacob fivo Pujare	102(e)/103(a) Urien	103(a) Jacob ivo Balabine

INTRODUCTION

The present invention relates to a lightweight end-to-end network architecture in which the transport layer is a Transmission Control Protocol (TCP) layer is disclosed. The network architecture also includes a Hyper Text Transport Protocol (HTTP) layer, a Messages over TCP (MT) protocol layer, a Message over Device-initiated HTTP (MDH) protocol layer, a multiplexing layer, and a facility layer. The MDH and MT layers are used in the alternative. The MT layer has a low overhead requirement. The MDH layer provides an enhanced firewall traversal capability.

//

Claims 1- 27 Rejection under 35 U.S.C. § 101

The Office at 3 states:

3. Claims 1-27 are rejected under 35 U.S.C. 101 because the claimed invention is directed to non-statutory subject matter. It was well-known in the network art that the first layer is the physical layer and the second layer is link layer. It was unclear if the applicant claimed the first layer is TCP and the second layer is HTTP then the network architecture, as claimed, without using the physical layer and link layer is impossible to communication. Examiner can not determine what applicant intended scope and Examiner can not determine without undue experimentation.

Claim 1 Rejection under 35 U.S.C. § 101

Applicant's claim 1 recites:

1. A computer network architecture comprising:
 - a first layer including a transmission control protocol connection;
 - a second layer including a hyper text transfer protocol connection built upon the first layer;
 - a first tunneling layer including a first tunneling protocol built upon the second layer to tunnel a message through the hyper text transfer protocol connection; and
 - a multiplexing layer to multiplex a plurality of messages for transmission through the first tunneling layer.

Firstly,

The Office states: "It was well-known in the network art that the first layer is the physical layer and the second layer is link layer."

Applicant submits that the Examiner is referring to an artificial model created for easy discussion rather than what Applicant has claimed.

The Examiner's reasoning is flawed as it arbitrarily makes distinctions and assumptions that are not relevant and are not what Applicant disclosed.

For example, the statement "...the network architecture, as claimed, without using the physical layer and link layer is impossible to communication." is irrelevant to what the Applicant has claimed. The Examiner could also argue "that without the physical connector to a communications channel, communication is impossible." While both statements may be true they are not germane to what Applicant has claimed.

Secondly,

One of skill in the network arts is familiar with both the ISO/OSI Network Model and the TCP/IP Network Model. As one of skill in the art is well aware, the TCP/IP Network Model has a link layer 1 on top of a network layer 2. The TCP/IP Network Model DOES NOT have a "physical layer". The Examiner's arguments with respect to a "physical layer" are immaterial to what the Applicant has claimed. Applicant is not claiming either of these models. Applicant submits that the Examiner is trying to "force fit" what Applicant has claimed into an existing network model to support a basis for rejection rather than understanding what the Applicant has claimed.

Thirdly,

Applicant submits that the Examiner is equating Applicant's "a first layer" to an OSI Layer 1. Applicant has never stated such in the application.

Fourthly,

Assuming arguendo that Applicant's "a first layer" is the same as OSI Layer 1, the argument makes no sense since Applicant clearly states "a first layer *including* a transmission control protocol ...". As the Examiner should be aware OSI Layer 1 is only a physical layer and does not include a TCP.

Fifthly,

Applicant submits that the Examiner is equating Applicant's "a second layer" to an OSI Layer 2. Applicant has never stated such in the application.

Sixthly,

Assuming arguendo that Applicant's "a second layer" is the same as OSI Layer 2, the argument makes no sense since Applicant clearly states "a second layer *including* a hyper text transfer protocol ...". As the Examiner should be aware OSI Layer 2 is only a data link layer and does not include a HTTP.

Seventhly,

The Office then states: "Examiner can not determine what applicant intended scope and Examiner can not determine without undue experimentation."

Applicant submits that the claims clearly define the invention and that the specification clearly describes and shows via the many figures the scope of the invention.

Eighthly,

Regarding the Office's contention "Examiner can not determine without undue experimentation." Applicant most strongly disagrees.

The specification and figures clearly show and describe embodiments of the invention as well as examples of usage. One skilled in the network arts can easily ascertain from, for example, Figure 2, the various layers and from the specification the interactions as well as the other Figures, such as Figure 5, the communications exchanges. There is no "undue experimentation" required to ascertain what the invention is or what Applicant has claimed.

In Summary – Claim 1

Applicant for the reasons detailed above submits that Applicant's claim 1 is directed to statutory matter. Applicant respectfully requests removal of the 35 U.S.C. § 101 rejection for independent claim 1 and allowance of claim 1.

Claims 2-9 Rejection under 35 U.S.C. § 101

Claims 2-9 are dependent on claim 1 and add additional limitations. Applicant submits that for the same reasons as detailed in Applicant's claim 1 discussion above, Applicant's claims 2-9 are directed to statutory matter and that the additional limitations of claims 2-9 are also directed to statutory matter. Applicant respectfully requests removal of the 35 U.S.C. § 101 rejection for claims 2-9 and allowance of claims 2-9.

Claim 10 Rejection under 35 U.S.C. § 101

Applicant's claim 10 recites:

10. A method for a computer network architecture comprising:

building a hyper text transfer protocol connection upon a transmission control protocol connection;

tunneling a message through the hyper text transfer protocol connection by using a first tunneling protocol layer including a first tunneling protocol; and

multiplexing a plurality of messages for transmission through the hyper text transfer protocol connection by using a multiplexing layer.

Applicant submits that for the same reasons as detailed above in the claim 1 discussion, independent claim 10 is directed to statutory matter. Applicant respectfully requests removal of the 35 U.S.C. § 101 rejection for independent claim 10 and allowance of claim 10.

Claims 11-18 Rejection under 35 U.S.C. § 101

Claims 11-18 are dependent on claim 10 and add additional limitations. Applicant submits that for the same reasons as detailed in Applicant's claim 10 discussion above, Applicant's claims 11-18 are directed to statutory matter and that the additional limitations of claims 11-18 are also directed to statutory matter. Applicant respectfully requests removal of the 35 U.S.C. § 101 rejection for claims 11-18 and allowance of claims 11-18.

Claim 19 Rejection under 35 U.S.C. § 101

Applicant's claim 19 recites:

19. A computer readable medium having instructions which, when executed by a processing system, cause the system to perform a method comprising:

building a hyper text transfer protocol connection upon a transmission control protocol connection;

tunneling a message through the Hyper Text Transfer Protocol connection by using a first tunneling protocol layer including a first tunneling protocol; and

multiplexing a plurality of messages for transmission through the Hyper Text Transfer Protocol connection by using a multiplexing layer.

Applicant submits that for the same reasons as detailed above in the claim 1 discussion, independent claim 19 is directed to statutory matter. Applicant respectfully

requests removal of the 35 U.S.C. § 101 rejection for independent claim 19 and allowance of claim 19.

Claims 20-27 Rejection under 35 U.S.C. § 101

Claims 20-27 are dependent on claim 19 and add additional limitations. Applicant submits that for the same reasons as detailed in Applicant's claim 19 discussion above, Applicant's claims 20-27 are directed to statutory matter and that the additional limitations of claims 20-27 are also directed to statutory matter. Applicant respectfully requests removal of the 35 U.S.C. § 101 rejection for claims 20-27 and allowance of claims 20-27.

//

Claim 1 Rejection under 35 U.S.C. § 102(e) - Bavadekar

The Office at 4 states:

4. As per claim 1, Bavadekar discloses a computer network architecture comprising:

a first layer including a transmission control protocol connection [Bavadekar, TCP connection, 0050];

a second layer including a hyper text transfer protocol connection built upon the first layer [Bavadelar, HTTP tunnel, 0050];

a first tunneling layer including a first tunneling protocol built upon the second layer to tunnel a message through the hyper text transfer protocol connection; and a multiplexing layer to multiplex a plurality of messages for transmission through the first tunneling layer [Bavadelar, HTTP tunnel may multiplex packets from the clients onto TCP connection, 0050].

The cited reference states:

[0050] In one embodiment, one Web server and one tunnel servlet may be used by two or more clients to communicate to a broker via tunnel connections. In this embodiment, the HTTP tunnel servlet may multiplex transport protocol packets from the two or more clients onto the TCP connection. In one embodiment, the tunnel servlet may extract the messaging system message information from received transport protocol packets and send only the message information to the broker over the TCP connection. In one embodiment, there may be one broker-side receive buffer for each tunnel connection. In another embodiment, a single receive buffer may be used for two or more tunnel connections.

Applicant's claim 1 recites:

1. A computer network architecture comprising:
 - a first layer including a transmission control protocol connection;
 - a second layer including a hyper text transfer protocol connection built upon the first layer;
 - a first tunneling layer including a first tunneling protocol built upon the second layer to tunnel a message through the hyper text transfer protocol connection; and
 - a multiplexing layer to multiplex a plurality of messages for transmission through the first tunneling layer.

To better assist in the discussion that follows, Applicant has added the **bolded** bracketed notations to the first portion of Bavadekar paragraph [0050] to better illustrate the entities as may be seen in Bavadekar Figure 3B and **bolded** bracketed notations to Applicant's claim 1.

[0050] In one embodiment, [see **Figure 3B**] one Web server [208] and one tunnel servlet [214] may be used by two or more clients [200A, 200B] to communicate to a broker [202] via tunnel connections. In this embodiment, the HTTP tunnel servlet [214] may multiplex transport protocol packets from the two or more clients [200A, 200B] onto the TCP connection [216]. In one embodiment, the tunnel servlet may extract the messaging system message information from received transport protocol packets and send only the message information to the broker over the TCP connection. In one embodiment, there may be one broker-side receive buffer for each tunnel

connection. In another embodiment, a single receive buffer may be used for two or more tunnel connections.

(Emphases added.)

Applicant's claim 1 recites:

1. A computer network architecture [see Figure 2] comprising:
 - a first layer including a transmission control protocol connection [222];
 - a second layer including a hyper text transfer protocol connection [232] built upon the first layer [222];
 - a first tunneling layer including a first tunneling protocol [240] built upon the second layer [232] to tunnel a message through the hyper text transfer protocol connection; and
 - a multiplexing layer [270] to multiplex a plurality of messages for transmission through the first tunneling layer [240].

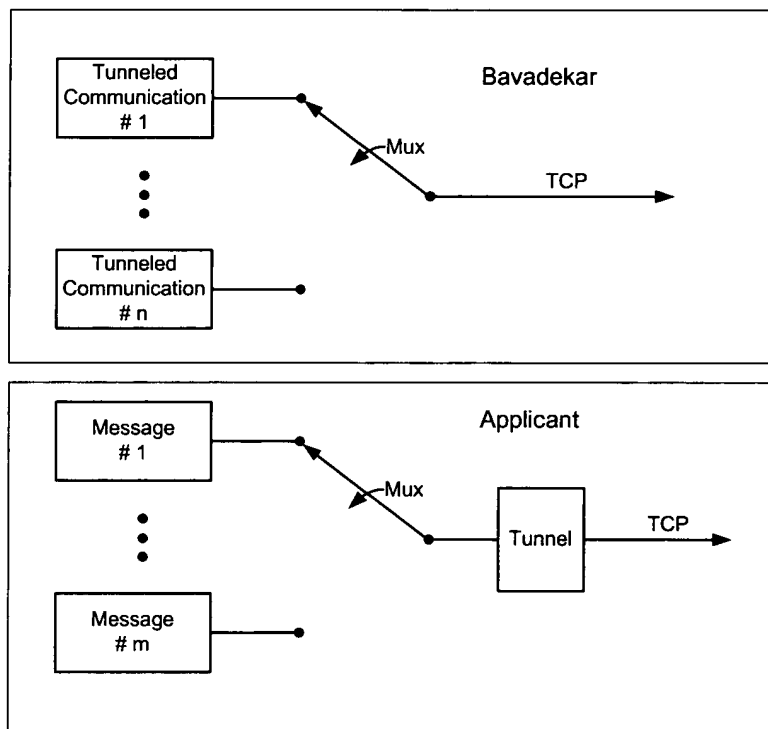
(Emphasis added.)

Applicant submits that Bavadekar additionally discloses at [0074]:

[0074] The HTTP tunnel client driver 220 may then send the messages as HTTP POST request payloads. The HTTP tunnel client driver 220 may also use separate HTTP requests to periodically pull any data sent by the other end of the connection. The HTTP requests may be sent through HTTP proxy 206, Internet 204, and firewall 210 to Web server 208. On Web server 208, the HTTP tunnel servlet 214 may act as a transceiver and may *multiplex the HTTP requests* from multiple clients *onto a single TCP connection* 216 with the broker 202. The HTTP tunnel broker driver 240 may receive the HTTP requests from the Web server 210 over TCP connection 216.

Thus Bavadekar discloses multiplexing the transport protocol packets [0050] or the HTTP requests [0074]. In either case the packets or requests are already arriving at the HTTP tunnel servlet 214 of Bavadekar as tunnel communications. Thus Bavadekar is multiplexing tunnel communications onto TCP 216. In contrast Applicant is multiplexing a plurality of messages into the tunneling which may then be transported via TCP.

The illustration below indicates the difference.



Multiplexing already tunneled communications onto TCP (Bavadekar) is not the same as multiplexing a plurality of messages into the tunneling which may then be transported via TCP (Applicant).

Further, Bavadekar then discusses at [0050], "In one embodiment, the tunnel servlet may extract the messaging system message information from received transport protocol packets and send only the message information to the broker over the TCP connection."

(Emphases added.)

This "teaches away" from the tunneling as disclosed by the Applicant.

Applicant submits that the Examiner has failed to establish a *prima facie* rejection under 35 U.S.C. § 102(e) because "Anticipation requires the presence in a single prior art reference disclosure of each and every element of the claimed invention, arranged as in the claim." *Lindemann Maschinenfabrik GMBH v. American Hoist & Derrick ("Lindemann")*, 730 F.2d 1452, 1458 (Fed. Cir. 1984) (emphasis added). Additionally, each and every element of the claim must be *exactly* disclosed in the anticipatory reference. *Titanium Metals Corp. of America v. Banner*, 778 F.2d 775, 777 (Fed. Cir. 1985).

Applicant submits that because Bavadekar fails to disclose multiplexing a plurality of messages for transmission through the first tunneling layer that Bavadekar cannot anticipate what Applicant has claimed. Applicant respectfully requests allowance of independent claim 1, and claims 2-9 which are dependent on claim 1.

Claim 2 Rejection under 35 U.S.C. § 102(e) - Bavadekar

The Office at 5 states:

5. As per claim 2, Bavadekar discloses the first tunneling protocol (i.e.: TCP) opens the HTTP connection between a server and a client [Bavadekar, 0050].

The cited reference states:

[0050] In one embodiment, one Web server and one tunnel servlet may be used by two or more clients to communicate to a broker via tunnel connections. In this embodiment, the HTTP tunnel servlet may multiplex transport protocol packets from the two or more clients onto the TCP connection. In one embodiment, the tunnel servlet may extract the messaging system message information from received transport protocol packets and send only the message information to the broker over the TCP connection. In one embodiment, there may be one broker-side receive buffer for each tunnel connection. In another embodiment, a single receive buffer may be used for two or more tunnel connections.

(Emphases added.)

Applicant's claim 2 recites:

2. The computer network architecture of claim 1, wherein the first tunneling protocol opens the hyper text transfer protocol connection between a server and a client.

Firstly,

Claim 2 is dependent on claim 1, and as discussed above in the claim 1 discussion, Bavadekar fails to disclose multiplexing a plurality of messages for transmission through the first tunneling layer, therefore Bavadekar cannot anticipate Applicant's claim 2. Further, Bavadekar cannot anticipate the further limitation of Applicant's claim 2. Applicant respectfully requests allowance of claim 2.

Secondly,

Applicant submits that the Office is incorrect when stating: "Bavadekar discloses the first tunneling protocol (i.e.: TCP)". (Emphasis added.) On the contrary, Bavadekar discloses only HTTP tunnel connections. Bavadekar does not disclose TCP to be a tunneling protocol. Bavadekar discloses TCP to be a connection (TCP Connection 216). The significance of this will become apparent in later claims where the Office tries to assert that the TCP is the first tunnel and that HTTP is a second tunnel.

Summary – claim 2

For the above reasons, Bavadekar cannot anticipate Applicant's claim 2. Applicant respectfully requests allowance of claim 2.

Claims 10-11, 19-20 Rejection under 35 U.S.C. § 102(e) - Bavadekar

The Office at 6 states:

6. Claims 10-11, 19-20 contain the similar limitations set forth of apparatus claims 1-2. Therefore, claims 10-11, 19-20 are rejected for the similar rationale set forth in claims 1-2.

Applicant submits that with respect to claims 10-11 and 19-20, that as discussed above in the claim 1-2 discussion, Bavadekar fails to disclose multiplexing a plurality of messages for transmission through the first tunneling layer, therefore Bavadekar cannot anticipate Applicant's claims 10-11 and 19-20. Applicant respectfully requests allowance of independent claims 10, and 19, and claims 11-18 and 20-27 which are dependent upon independent claims 10 and 19 respectively.

Claim 3 Rejection under 35 U.S.C. § 102(e) - Bavadekar

The Office at 7 states:

7. As per claim 3, Bavadekar discloses a second tunneling layer (i.e.: HTTP tunnel) including a second tunneling protocol (i.e.: HTTP protocol) built upon the first layer to tunnel a message through the TCP connection [Bavadekar, 0050].
(Emphasis added.)

The cited reference states in part: [Bavadekar, 0050].

[0050] In one embodiment, one Web server and one tunnel servlet may be used by two or more clients to communicate to a broker via tunnel connections. In this embodiment, the HTTP tunnel

servlet may multiplex transport protocol packets from the two or more clients onto the TCP connection. In one embodiment, the tunnel servlet may extract the messaging system message information from received transport protocol packets and send only the message information to the broker over the TCP connection. In one embodiment, there may be one broker-side receive buffer for each tunnel connection. In another embodiment, a single receive buffer may be used for two or more tunnel connections.

Applicant's claim 3 recites:

3. The computer network architecture of claim 1, further comprising:
a second tunneling layer including a second tunneling protocol built upon the first layer to tunnel a message through the transmission control protocol connection.

Applicant submits that Bavadekar does not disclose any second tunneling layer. The Office states: "Bavadekar discloses a second tunneling layer (i.e.: HTTP tunnel) including a second tunneling protocol (i.e.: HTTP protocol) built upon the first layer to tunnel a message through the TCP connection [Bavadekar, 0050]."

(Emphasis added.)

Firstly,

Claim 3 is dependent on claim 1, and as discussed above in the claim 1 discussion, Bavadekar fails to disclose multiplexing a plurality of messages for transmission through the first tunneling layer, therefore Bavadekar cannot anticipate Applicant's claim 3. Further, Bavadekar cannot anticipate the further limitation of Applicant's claim 3. Applicant respectfully requests allowance of claim 3.

Secondly,

As explained above (claim 2 – Secondly), Bavadekar discloses only a single tunnel, that of the HTTP tunnel connections. Bavadekar does not disclose TCP to be a tunneling protocol. Bavadekar discloses TCP to be a connection (TCP Connection 216). Thus, Bavadekar does not disclose what Applicant has claimed (“a second tunneling layer”).

Thirdly,

The Office statement is inconsistent. On the one hand, the Office is claiming the TCP to be a first tunnel, HTTP to be the second tunnel, and then wants to assert “to tunnel a message through the TCP connection.” (Emphasis added.) The Office cannot have the TCP as both a connection and tunnel. Since Bavadekar describes the TCP as a connection, Applicant submits that the Office is incorrect in characterizing the TCP as a tunnel. Reference is made to Bavadekar paragraph [0036] where tunneling is defined.

Summary – claim 3

For the above reasons, Bavadekar cannot anticipate Applicant's claim 3. Applicant respectfully requests allowance of claim 3, and claims 4-5 which are dependent on claim 3.

Claim 4 Rejection under 35 U.S.C. § 102(e) - Bavadekar

The Office at 8 states:

8. As per claim 4, Bavadekar-Pujare disclose the second tunneling protocol is used to open the TCP connection between the server and the client [Bavadekar, 0050].

The cited reference states:

[0050] In one embodiment, one Web server and one tunnel servlet may be used by two or more clients to communicate to a broker via tunnel connections. In this embodiment, the HTTP tunnel servlet may multiplex transport protocol packets from the two or more clients onto the TCP connection. In one embodiment, the tunnel servlet may extract the messaging system message information from received transport protocol packets and send only the message information to the broker over the TCP connection. In one embodiment, there may be one broker-side receive buffer for each tunnel connection. In another embodiment, a single receive buffer may be used for two or more tunnel connections.

Applicant's claim 4 recites:

4. The computer network architecture of claim 3, wherein the second tunneling protocol is used to open the transmission control protocol connection between the server and the client.

Firstly,

Claim 4 is dependent on claim 3 which is dependent on claim 1, and as discussed above in the claim 1 discussion, Bavadekar fails to disclose multiplexing a plurality of messages for transmission through the first tunneling layer, therefore Bavadekar cannot anticipate Applicant's claim 4. Further, Bavadekar cannot anticipate the further limitation of Applicant's claims 3 and 4. Applicant respectfully requests allowance of claim 4, and claim 5 which is dependent on claim 4.

Secondly,

Nowhere in the cited section does Bavadekar disclose or discuss a second tunneling protocol as Applicant has claimed. Bavadekar only discusses a single tunnel protocol, that being an HTTP tunnel protocol (see Bavadekar Abstract – “A system and method for providing HTTP tunnel connections between entities such as clients and servers in a messaging system is described.” (Emphasis added.) Since Bavadekar does not disclose a second tunneling protocol as Applicant has claimed, Bavadekar cannot anticipate Applicant's claim.

Thirdly,

Applicant notes that the Office mentions Pujare in paragraph 8. but cites no reference or argument. Applicant submits that Pujare does not anticipate the limitations of claim 1 or claim 4.

Summary – claim 4

For the above reasons, Bavadekar cannot anticipate Applicant's claim 4. Applicant respectfully requests allowance of claim 4, and claim 5 which is dependent on claim 4.

Claim 5 Rejection under 35 U.S.C. § 102(e) - Bavadekar

The Office at 9 states:

9. As per claim 5, Bavadekar-Pujare disclose tunneling protocol opens the HTTP connection if the second tunneling protocol is not successful in opening the TCP connection [Bavadekar, 0050].

The cited reference states:

[0050] In one embodiment, one Web server and one tunnel servlet may be used by two or more clients to communicate to a broker via tunnel connections. In this embodiment, the HTTP tunnel servlet may multiplex transport protocol packets from the two or more clients onto the TCP connection. In one embodiment, the tunnel servlet may extract the messaging system message information from received transport protocol packets and send only the message information to the broker over the TCP connection. In one embodiment, there may be one broker-side receive buffer for

each tunnel connection. In another embodiment, a single receive buffer may be used for two or more tunnel connections.

Applicant's claim 5 recites:

5. The computer network architecture of claim 4, wherein the first tunneling protocol opens the hyper text transfer protocol connection if the second tunneling protocol is not successful in opening the transmission control protocol connection.

Firstly,

Claim 5 is dependent on claim 4 which is dependent on claim 3 which is dependent on claim 1, and as discussed above in the claim 1 discussion, Bavadekar fails to disclose multiplexing a plurality of messages for transmission through the first tunneling layer, therefore Bavadekar cannot anticipate Applicant's claim 5. Further, Bavadekar cannot anticipate the further limitation of Applicant's claims 3, 4, and 5. Applicant respectfully requests allowance of claim 5.

Secondly,

Nowhere in the cited section does Bavadekar disclose or discuss the condition where another protocol is tried if a first one is not successful (i.e. first protocol is attempted if a second protocol is not successful). Since Bavadekar does not disclose this limitation of Applicant's claim 5, Bavadekar cannot anticipate Applicant's claim.

Thirdly,

Applicant notes that the Office mentions Pujare in paragraph 9. but cites no reference or argument. Applicant submits that Pujare does not anticipate the limitations of claim 1 or claim 5.

Summary – claim 5

For the above reasons, Bavadekar cannot anticipate Applicant's claim 5. Applicant respectfully requests allowance of claim 5.

Claim 6 Rejection under 35 U.S.C. § 102(e) - Bavadekar

The Office at 10 states:

10. As per claim 6, Bavadekar-Pujare disclose the messages include binary format [Pujare, digital signals, 0155].

Applicant notes that at this point, it is unclear as to whether the Office is referring to the Bavadekar reference mentioned or to Pujare, therefore Applicant will discuss both.

The cited references state:

(Pujare) [0155] E) Network Spoofing for Client-server Applications

(Bavadekar) [0155] Various embodiments may further include receiving, sending or storing instructions and/or data implemented in accordance with the foregoing description upon a carrier

medium. Generally speaking, a carrier medium may include storage media or memory media such as magnetic or optical media, e.g., disk or CD-ROM, volatile or non-volatile media such as RAM (e.g. SDRAM, DDR SDRAM, RDRAM, SRAM, etc.), ROM, etc. as well as transmission media or signals such as electrical, electromagnetic, or digital signals, conveyed via a communication medium such as network and/or a wireless link.

Applicant's claim 6 recites:

6. The computer network of claim 1, wherein the messages include binary format messages.

Firstly,

With respect to Bavadekar:

Claim 6 is dependent on claim 1, and as discussed above in the claim 1 discussion, Bavadekar fails to disclose multiplexing a plurality of messages for transmission through the first tunneling layer, therefore Bavadekar cannot anticipate Applicant's claim 6. Further, Bavadekar cannot anticipate the further limitation of Applicant's claim 6. Applicant respectfully requests allowance of claim 6.

Secondly,

With respect to Pujare:

Pujare at the cited section only discloses "Network Spoofing for Client-server Applications" and does not disclose multiplexing a plurality of messages for transmission

through the first tunneling layer which Applicant has claimed in claim 1. Therefore Pujare cannot anticipate Applicant's claim 1 upon which claim 6 is dependent and cannot anticipate Applicant's claim 6. Further, Pujare fails to disclose or suggest the further limitations of claim 6. Applicant respectfully requests allowance of claim 6.

Summary – claim 6

For the above reasons, neither Bavadekar nor Pujare anticipate Applicant's claim 6. Applicant respectfully requests allowance of claim 6.

Claim 7 Rejection under 35 U.S.C. § 102(e) - Bavadekar

The Office at 11 states:

11. As per claim 7, Bavadekar-Pujare disclose the plurality of messages includes a plurality of operational messages and a plurality of administrative messages [Bavadekar, administrative control, 0038].

The cited reference states:

[0038] In an enterprise that uses the Internet, a proxy server is a server that acts as an intermediary between a workstation user and the Internet so that the enterprise can ensure security, administrative control, and caching service. A proxy server may be associated with or part of a gateway server that separates the enterprise network from the outside network and a firewall server that protects the enterprise network from outside intrusion.

(Emphasis added.)

Applicant's claim 7 recites:

7. The computer network architecture of claim 1, wherein the plurality of messages includes a plurality of operational messages and a plurality of administrative messages.

Firstly,

Claim 7 is dependent on claim 1, and as discussed above in the claim 1 discussion, Bavadekar fails to disclose multiplexing a plurality of messages for transmission through the first tunneling layer, therefore Bavadekar cannot anticipate Applicant's claim 7. Further, Bavadekar cannot anticipate the further limitation of Applicant's claim 7. Applicant respectfully requests allowance of claim 7.

Secondly,

Applicant submits that the Office is incorrectly equating Bavadekar's "administrative control" with Applicant's "administrative messages". Control is not the same as messages.

Thirdly,

Applicant notes that the Office mentions Pujare in paragraph 11. but cites no reference or argument. Applicant submits that Pujare does not anticipate the limitations of claim 1 or claim 7.

Summary – claim 7

For the above reasons, Bavadekar does not anticipate Applicant's claim 7. Applicant respectfully requests allowance of claim 7, and claims 8-9 which are dependent on claim 7.

Claim 8 Rejection under 35 U.S.C. § 102(e) - Bavadekar

The Office at 12 states:

12. As per claim 8, Bavadekar-Pujare disclose the operational messages include operational data [Pujare, parameter, 0148].

Applicant again notes that at this point, it is unclear as to whether the Office is referring to the Bavadekar reference mentioned or to Pujare, therefore Applicant will discuss both.

The cited references state:

(Pujare) [0148] i) Client License Manager 303--This is the same component explained above.

(Bavadekar) [0148] Either client 200 or server 202 may set connection options. This may be used to control the runtime parameters associated with a connection. For example, this may be used to control how often packets are pulled from a Web server. As another example, this may be used to

configure how long a client may remain inactive before the connection is dropped. The connection option values are part of the connection state information, and hence, in one embodiment, an HTTP packet including the following information may be used to update the connection options:

Applicant's claim 8 recites:

8. The computer network architecture of claim 7, wherein the operational messages include operational data.

Firstly,

With regard to Bavadekar:

Claim 8 is dependent on claim 7 which is dependent on claim 1, and as discussed above in the claim 1 discussion, Bavadekar fails to disclose multiplexing a plurality of messages for transmission through the first tunneling layer, therefore Bavadekar cannot anticipate Applicant's claim 8. Further, Bavadekar cannot anticipate the further limitations of Applicant's claims 7 and 8. Applicant respectfully requests allowance of claim 7.

Secondly,

With respect to Pujare:

Pujare at the cited section only discloses "Client License Manager 303--This is the same component explained above." and does not disclose multiplexing a plurality of messages for transmission through the first tunneling layer which Applicant has claimed in claim 1.

Therefore Pujare cannot anticipate Applicant's claim 1 upon which claim 8 is dependent and

cannot anticipate Applicant's claim 8. Further, Pujare fails to disclose or suggest the further limitations of claims 7 and 8. Applicant respectfully requests allowance of claim 8.

Summary – claim 8

For the above reasons, neither Bavadekar nor Pujare anticipate Applicant's claim 8. Applicant respectfully requests allowance of claim 8.

Claim 9 Rejection under 35 U.S.C. § 102(e) - Bavadekar

The Office at 13 states:

13. As per claim 9, Bavadekar-Pujare disclose the administrative messages can be selected from the group consisting of debug messages, firmware update messages and parameter configuration messages [Pujare, update options, parameter, 0148].

Applicant again notes that at this point, it is unclear as to whether the Office is referring to the Bavadekar reference mentioned or to Pujare, therefore Applicant will discuss both.

The cited references state:

(Pujare) [0148] i) Client License Manager 303--This is the same component explained above.

(Bavadekar) [0148] Either client 200 or server 202 may set connection options. This may be used to control the runtime parameters associated with a connection. For example, this may be used to control how often packets are pulled from a Web server. As another example, this may be used to configure how long a client may remain inactive before the connection is dropped. The connection option values are part of the connection state information, and hence, in one embodiment, an HTTP packet including the following information may be used to update the connection options:

Applicant's claim 9 recites:

9. The computer network architecture of claim 7, wherein the administrative messages can be selected from the group consisting of debug messages, firmware update messages and parameter configuration messages.

Firstly,

With regard to Bavadekar:

Claim 9 is dependent on claim 7 which is dependent on claim 1, and as discussed above in the claim 1 discussion, Bavadekar fails to disclose multiplexing a plurality of messages for transmission through the first tunneling layer, therefore Bavadekar cannot anticipate Applicant's claim 9. Further, Bavadekar cannot anticipate the further limitations of Applicant's claims 7 and 9. Applicant respectfully requests allowance of claim 9.

Secondly,

With respect to Pujare:

Pujare at the cited section only discloses "Client License Manager 303--This is the same component explained above." and does not disclose multiplexing a plurality of messages for transmission through the first tunneling layer which Applicant has claimed in claim 1. Therefore Pujare cannot anticipate Applicant's claim 1 upon which claim 9 is dependent and cannot anticipate Applicant's claim 9. Further, Pujare fails to disclose or suggest the further limitations of claims 7 and 9. Applicant respectfully requests allowance of claim 9.

Summary – claim 9

For the above reasons, neither Bavadekar nor Pujare anticipate Applicant's claim 9. Applicant respectfully requests allowance of claim 9.

//

Claims 12-18 and 21-27 Rejection under 35 U.S.C. § 102(e) - Bavadekar

Applicant notes that while claims 12-18 and 21-27 stand rejected under 35 U.S.C. § 102(e) as being anticipated by Bavadekar that the Office has failed to particularly address each of the claims.

Applicant submits that for substantially the same reasons as detailed above in the claims 1-9 discussions that claims 12-18 and 21-27 are not anticipated by the cited art. Applicant respectfully requests allowance of claims 12-18 and 21-27.

CONCLUSION

Applicant submits that the rejection of dependent claims not specifically addressed, are addressed by Applicant's arguments to the independent claims on which they depend.

Applicant respectfully submits that the appealed claims in this application are patentable, and requests that the Board of Patent Appeals and Interferences direct allowance of all claims.

Respectfully submitted,

Heimlich Law

11/29/2006

Date

Alan Heimlich / Reg 48808

Attorney for Applicant(s)

Customer No. 40418

5952 Dial Way
San Jose, CA 95129

Tel: 408 253-3860
Eml: alanheimlich@heimlichlaw.com

VIII. CLAIMS APPENDIX

The claims involved in this appeal (all pending claims) are as follows:

CLAIMS

What is claimed is:

1. (original) A computer network architecture comprising:
 - a first layer including a transmission control protocol connection;
 - a second layer including a hyper text transfer protocol connection built upon the first layer;
 - a first tunneling layer including a first tunneling protocol built upon the second layer to tunnel a message through the hyper text transfer protocol connection; and
 - a multiplexing layer to multiplex a plurality of messages for transmission through the first tunneling layer.
2. (original) The computer network architecture of claim 1, wherein the first tunneling protocol opens the hyper text transfer protocol connection between a server and a client.
3. (original) The computer network architecture of claim 1, further comprising:
 - a second tunneling layer including a second tunneling protocol built upon the first layer to tunnel a message through the transmission control protocol connection.

4. (original) The computer network architecture of claim 3, wherein the second tunneling protocol is used to open the transmission control protocol connection between the server and the client.
5. (original) The computer network architecture of claim 4, wherein the first tunneling protocol opens the hyper text transfer protocol connection if the second tunneling protocol is not successful in opening the transmission control protocol connection.
6. (original) The computer network of claim 1, wherein the messages include binary format messages.
7. (original) The computer network architecture of claim 1, wherein the plurality of messages includes a plurality of operational messages and a plurality of administrative messages.
8. (original) The computer network architecture of claim 7, wherein the operational messages include operational data.
9. (original) The computer network architecture of claim 7, wherein the administrative messages can be selected from the group consisting of debug messages, firmware update messages and parameter configuration messages.
10. (original) A method for a computer network architecture comprising:

building a hyper text transfer protocol connection upon a transmission control protocol connection;

tunneling a message through the hyper text transfer protocol connection by using a first tunneling protocol layer including a first tunneling protocol; and

multiplexing a plurality of messages for transmission through the hyper text transfer protocol connection by using a multiplexing layer.

11. (original) The method of claim 10, wherein opening the hyper text transfer protocol connection between a server and a client by using the first tunneling layer.

12. (original) The method of claim 10, further comprising:

tunneling a message through the transmission control protocol connection by using a second tunneling protocol layer including a second tunneling protocol built upon the transmission control protocol connection.

13. (original) The method of claim 12, wherein opening the transmission control protocol connection between a server and a client by using the second tunneling protocol.

14. (original) The method of claim 13, wherein opening the hyper text transfer protocol connection by using the first tunneling protocol if the transmission control protocol connection is not successfully opened by using the second tunneling protocol.

15. (original) The method of claim 10, wherein the messages include binary format messages.

16. (original) The method of claim 10, wherein the plurality of messages include a plurality of operational messages and a plurality of administrative messages.

17. (original) The method of claim 16, wherein the operational messages include operational data.

18. (original) The method of claim 16, wherein the administrative messages can be selected from the group consisting of debug messages, firmware update messages and parameter configuration messages.

19. (original) A computer readable medium having instructions which, when executed by a processing system, cause the system to perform a method comprising:

building a hyper text transfer protocol connection upon a transmission control protocol connection;

tunneling a message through the Hyper Text Transfer Protocol connection by using a first tunneling protocol layer including a first tunneling protocol; and

multiplexing a plurality of messages for transmission through the Hyper Text Transfer Protocol connection by using a multiplexing layer.

20. (original) The medium of claim 19, wherein opening the Hyper Text Transfer Protocol connection between a server and a client by using the first tunneling layer.

21. (original) The medium of claim 19, further comprising:

tunneling a message through the transmission control protocol connection by using a second tunneling protocol layer including a second tunneling protocol built upon the Transmission Control Protocol Connection.

22. (original) The medium of claim 21, wherein opening the Transmission Control Protocol connection between a server and a client by using the second tunneling protocol.

23. (original) The medium of claim 22, wherein opening the Hyper Text Transfer Protocol connection by using the first tunneling protocol if the Transmission Control Protocol connection is not successfully opened by using the second tunneling protocol.

24. (original) The medium of claim 19, wherein the messages include binary format messages.

25. (original) The medium of claim 19, wherein the plurality of messages include a plurality of operational messages and a plurality of administrative messages.

26. (original) The medium of claim 25, wherein the operational messages include operational data.

27. (original) The medium of claim 25, wherein the administrative messages can be selected from the group consisting of debug messages, firmware update messages and parameter configuration messages.

IX. EVIDENCE APPENDIX

Grouping of any Evidence for Claim purposes is for the convenience of reduced duplication and is NOT to be interpreted as the Grouping of Claims for Arguments under 37 C.F.R. § 41.37.

(A) Evidence for Claims 1-27 – Relied Upon

The following item (1) listed below is hereby entered as evidence relied upon by the Examiner as to grounds of rejection for claims 1-27, to be reviewed on appeal. Also listed for each item is where said evidence was entered into the record by the Examiner.

(1) Copy of US Patent Application Number 20030009571 ("Bavadekar"). This evidence was entered into the record by the Examiner on page 3 paragraph 3. of the Office Action mailed 03/23/2006.

Copies of all References follows.

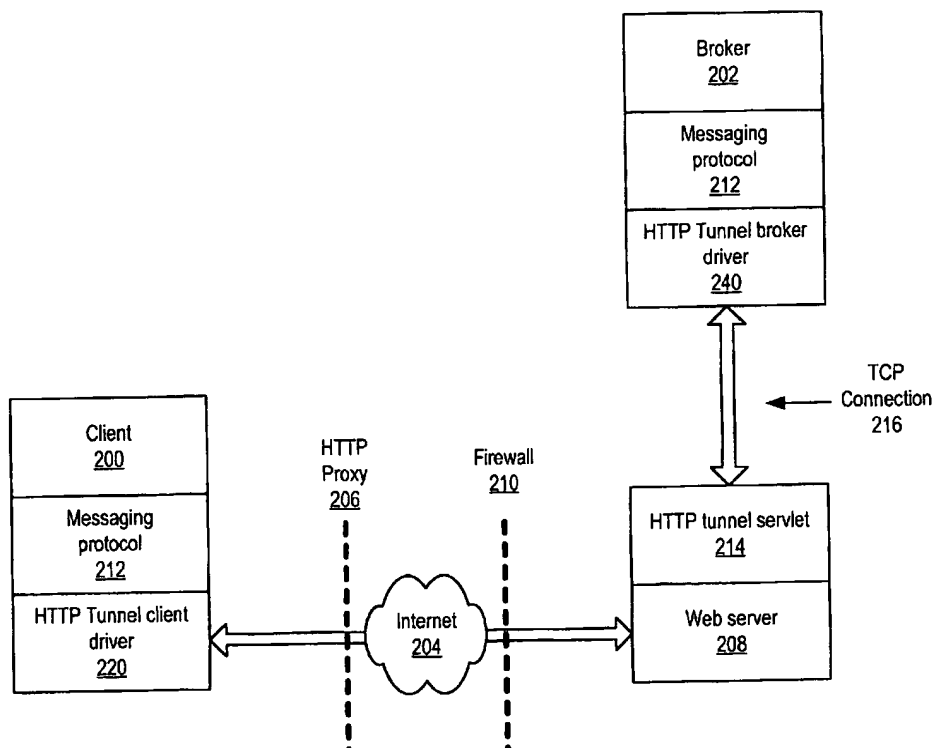
//



US 20030009571A1

(19) **United States**(12) **Patent Application Publication**
Bavadekar(10) **Pub. No.: US 2003/0009571 A1**(43) **Pub. Date: Jan. 9, 2003**(54) **SYSTEM AND METHOD FOR PROVIDING
TUNNEL CONNECTIONS BETWEEN
ENTITIES IN A MESSAGING SYSTEM**(76) **Inventor: Shailesh S. Bavadekar, Sunnyvale, CA
(US)****Correspondence Address:**
Robert C. Kowert
Conley, Rose, & Tayon, P.C.
P.O. Box 398
Austin, TX 78767-1400 (US)(21) **Appl. No.: 09/894,318**(22) **Filed: Jun. 28, 2001****Publication Classification**(51) **Int. Cl.⁷ G06F 15/16**(52) **U.S. Cl. 709/230; 709/203**(57) **ABSTRACT**

A system and method for providing HTTP tunnel connections between entities such as clients and servers in a messaging system is described. An HTTP tunnel connection layer is described that may be used to provide reliable, full duplex virtual connections between entities (e.g. clients and brokers) in a distributed application environment using a messaging system. Also described is a novel HTTP tunneling protocol that may be used by the HTTP tunnel connection layer. The HTTP tunnel connection layer may be used by clients to access messaging servers through proxy servers and firewalls, thus expanding the scope of from where clients can access brokers. Using this layer, brokers as well as clients may initiate messaging system messages. This layer may also provide guaranteed data delivery with correct sequencing even in case of a failure on the network. This layer may also provide end-to-end flow control.



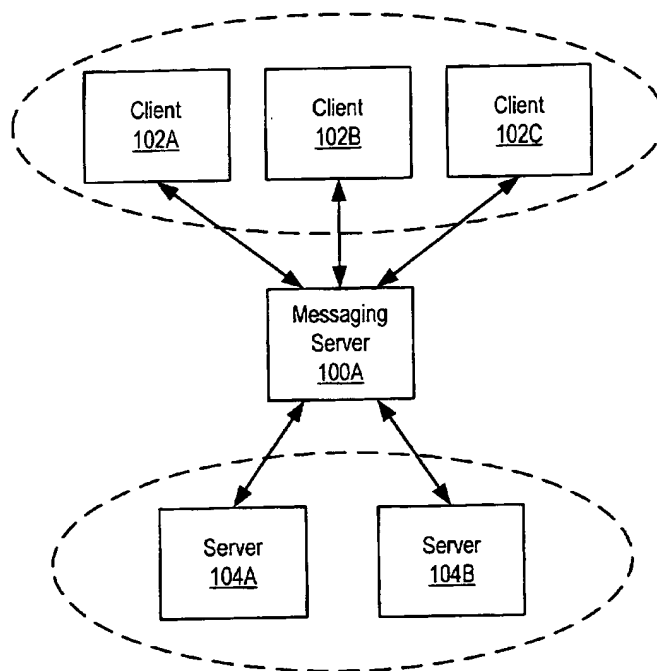


Figure 1 - Prior Art

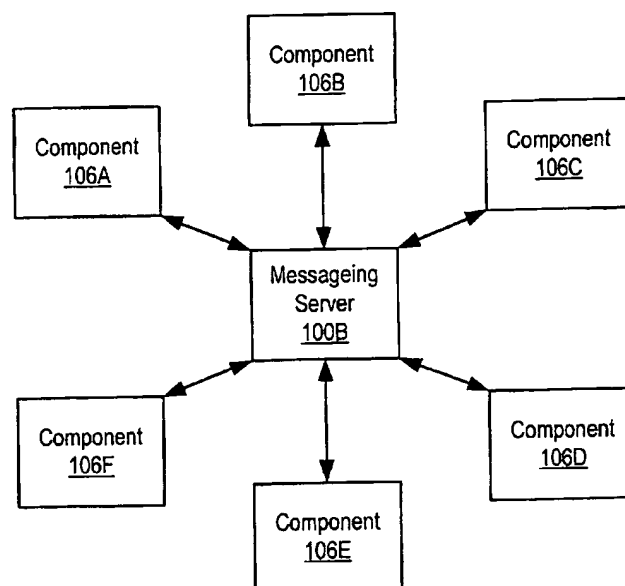


Figure 2 - Prior Art

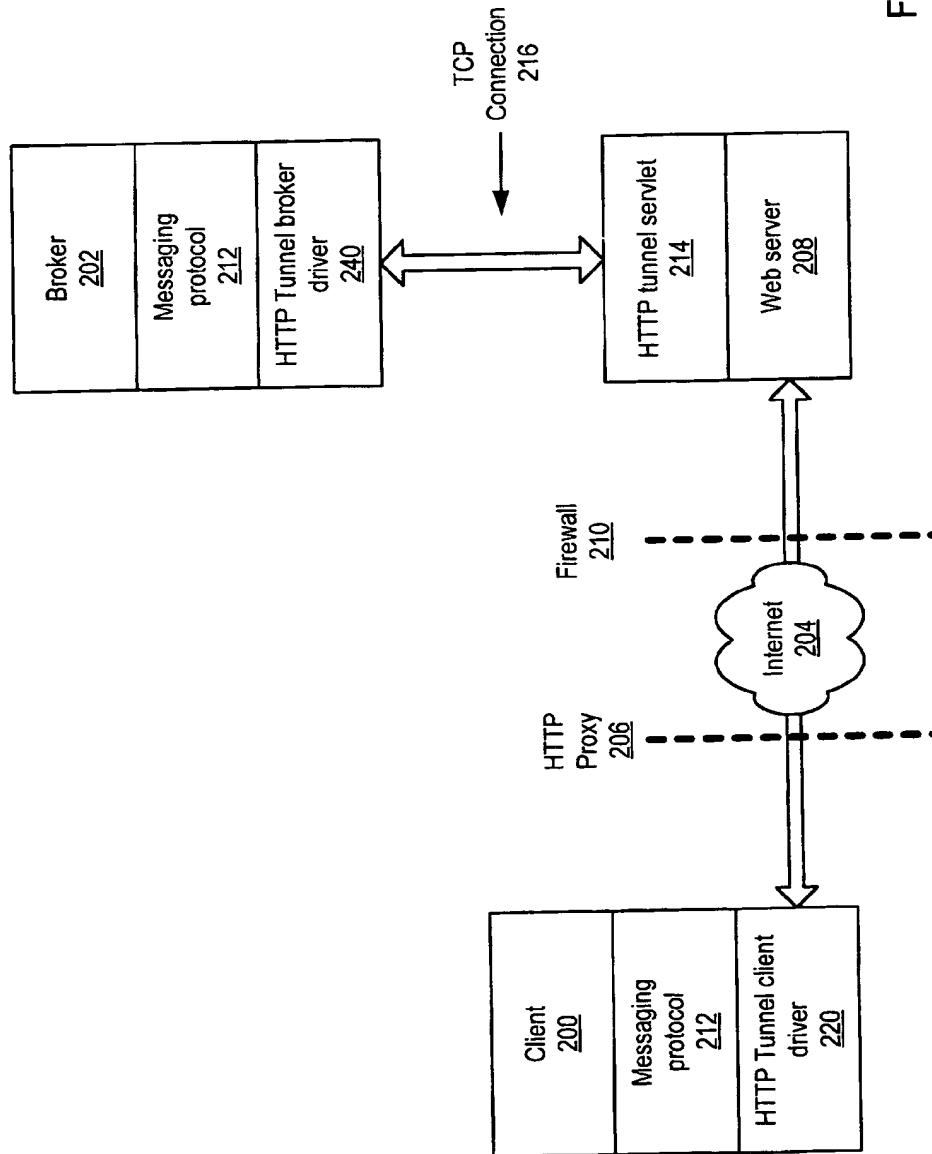


Figure 3A

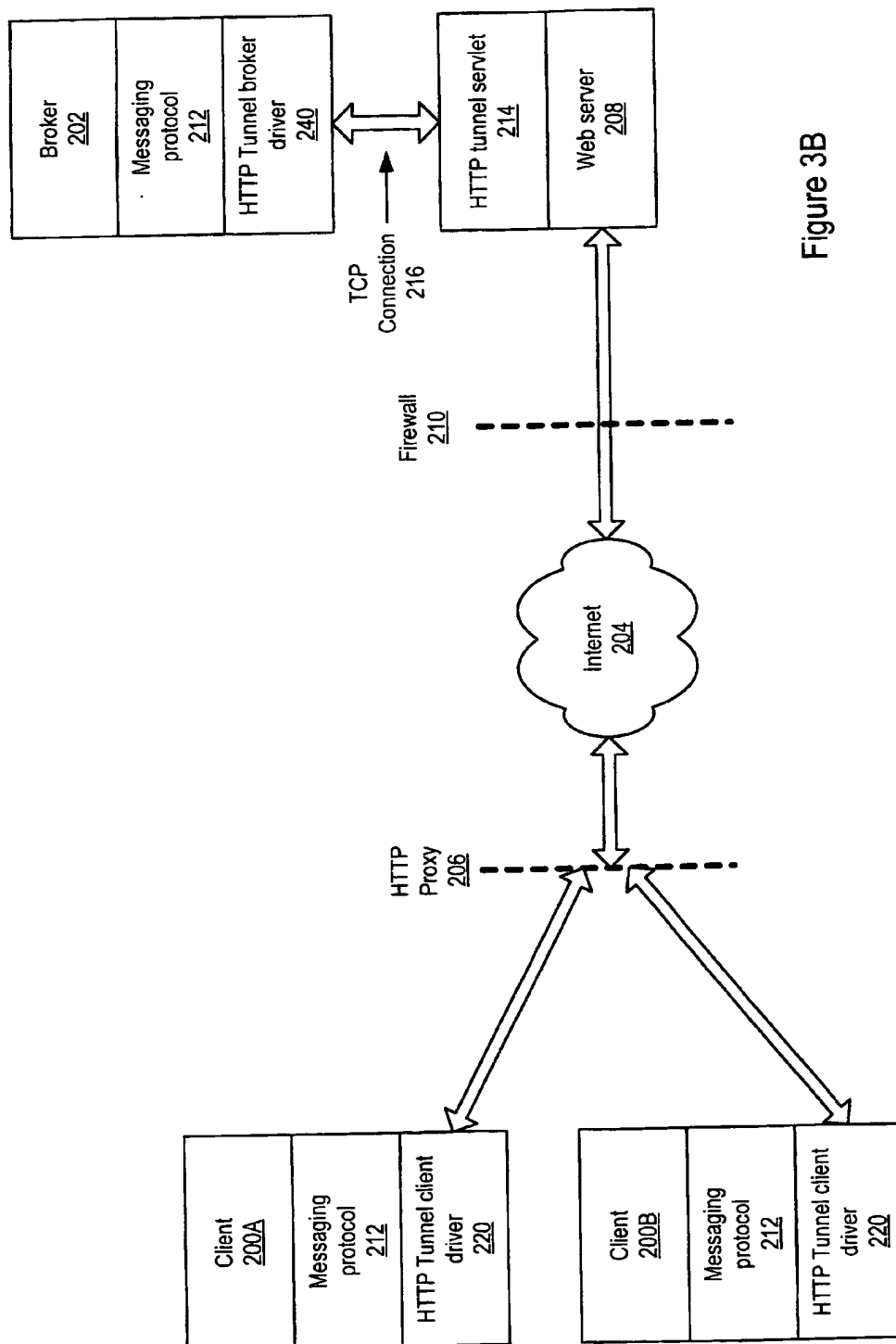


Figure 3B

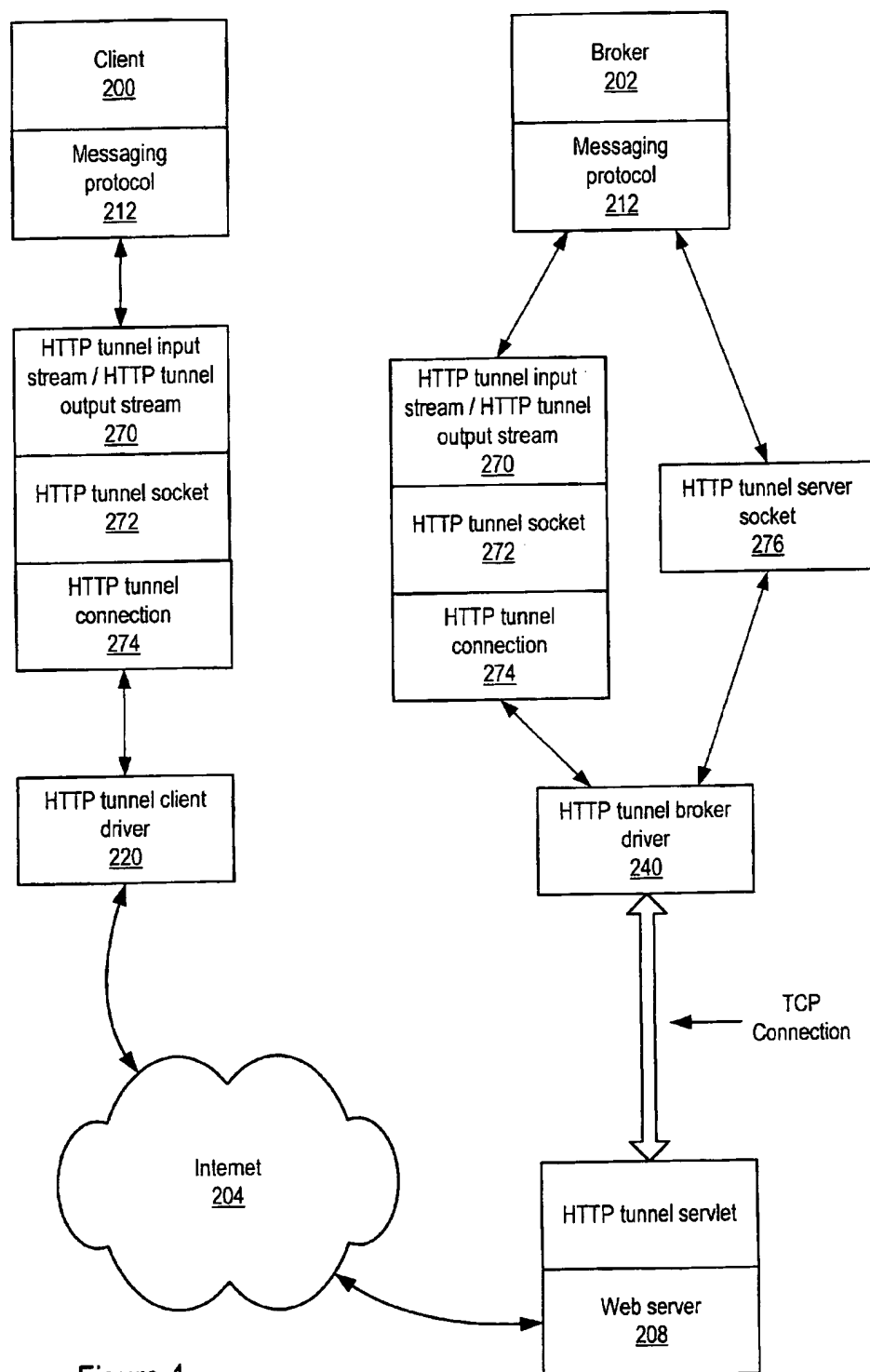


Figure 4

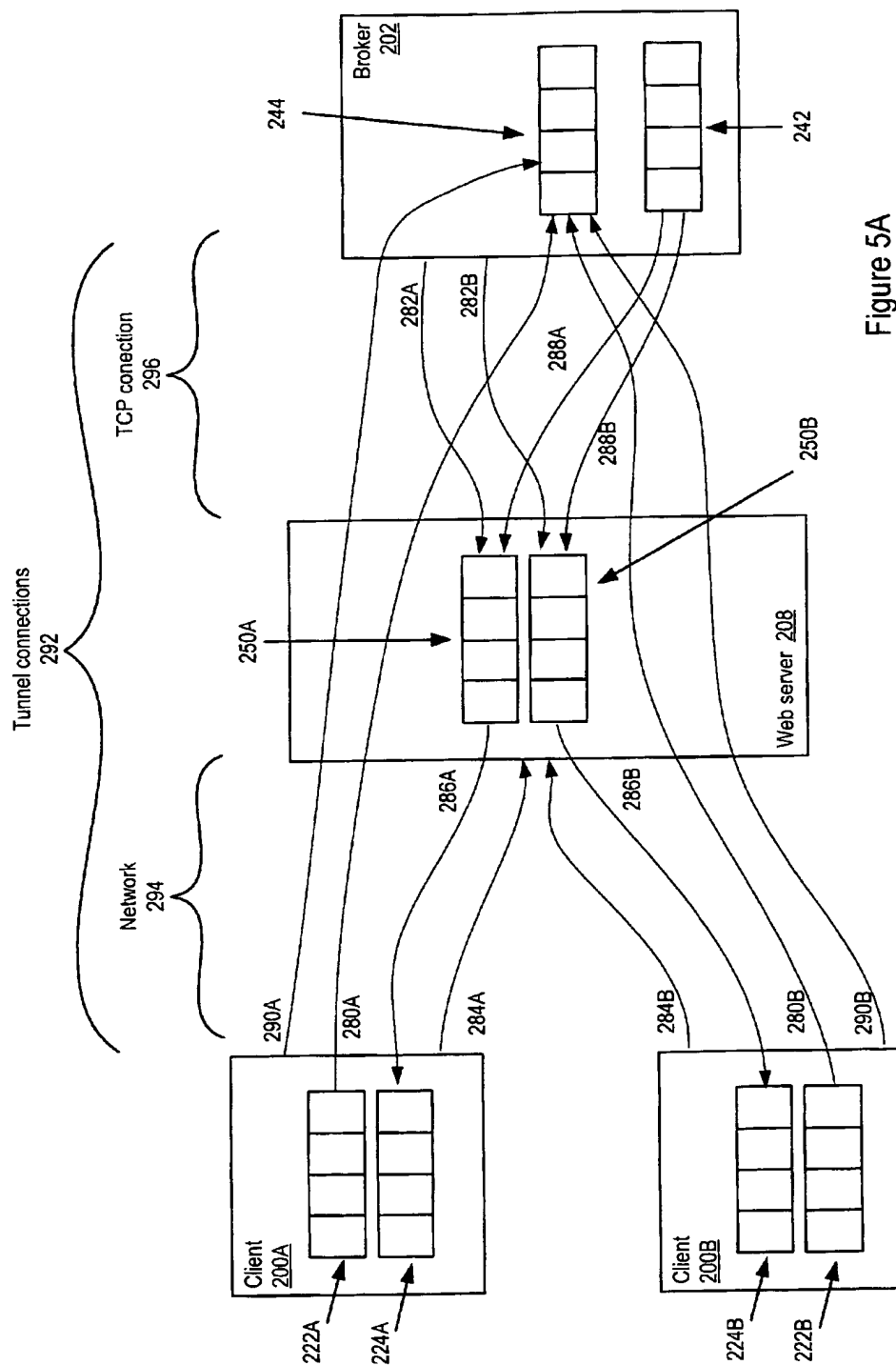


Figure 5A

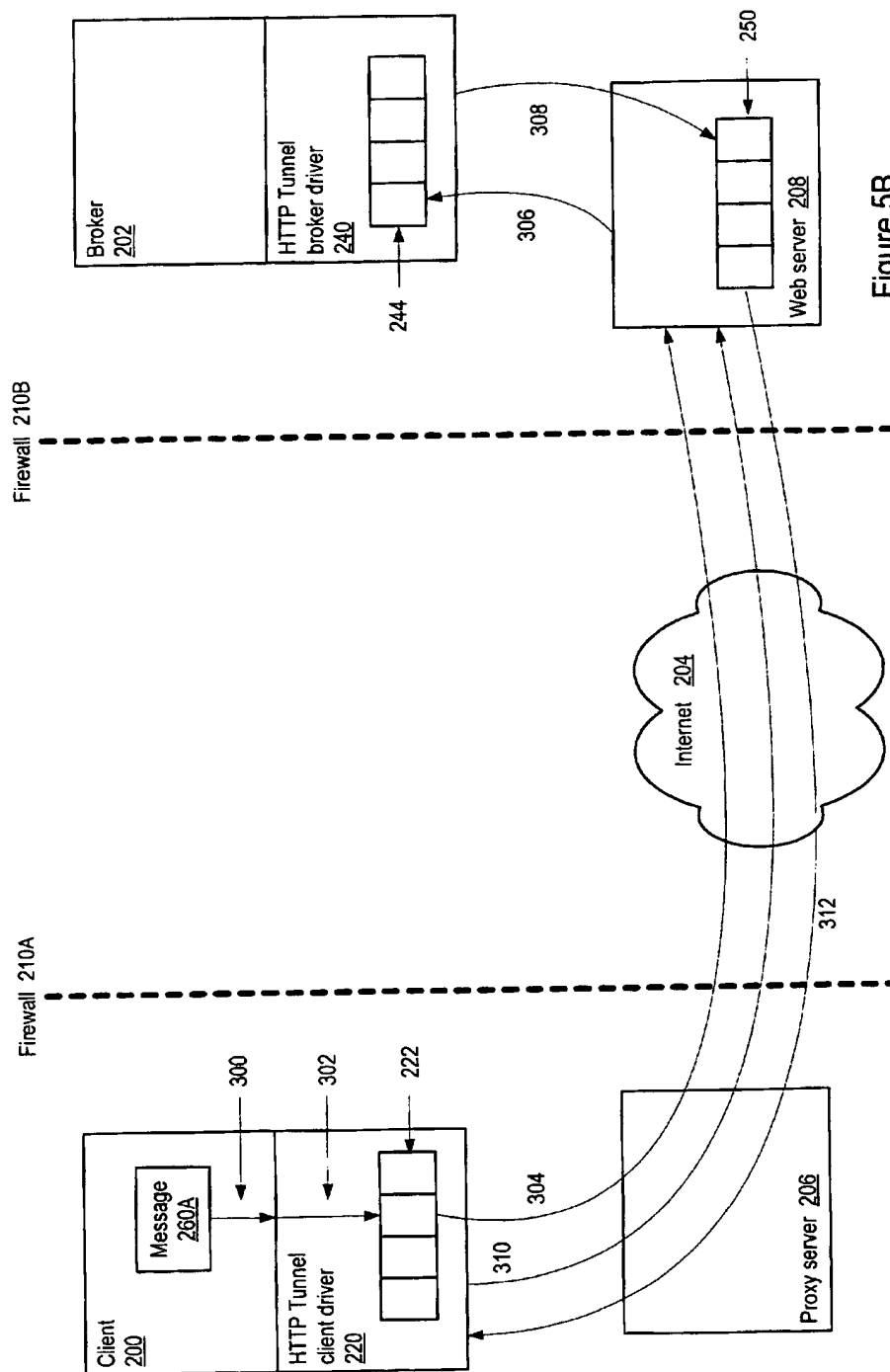


Figure 5B

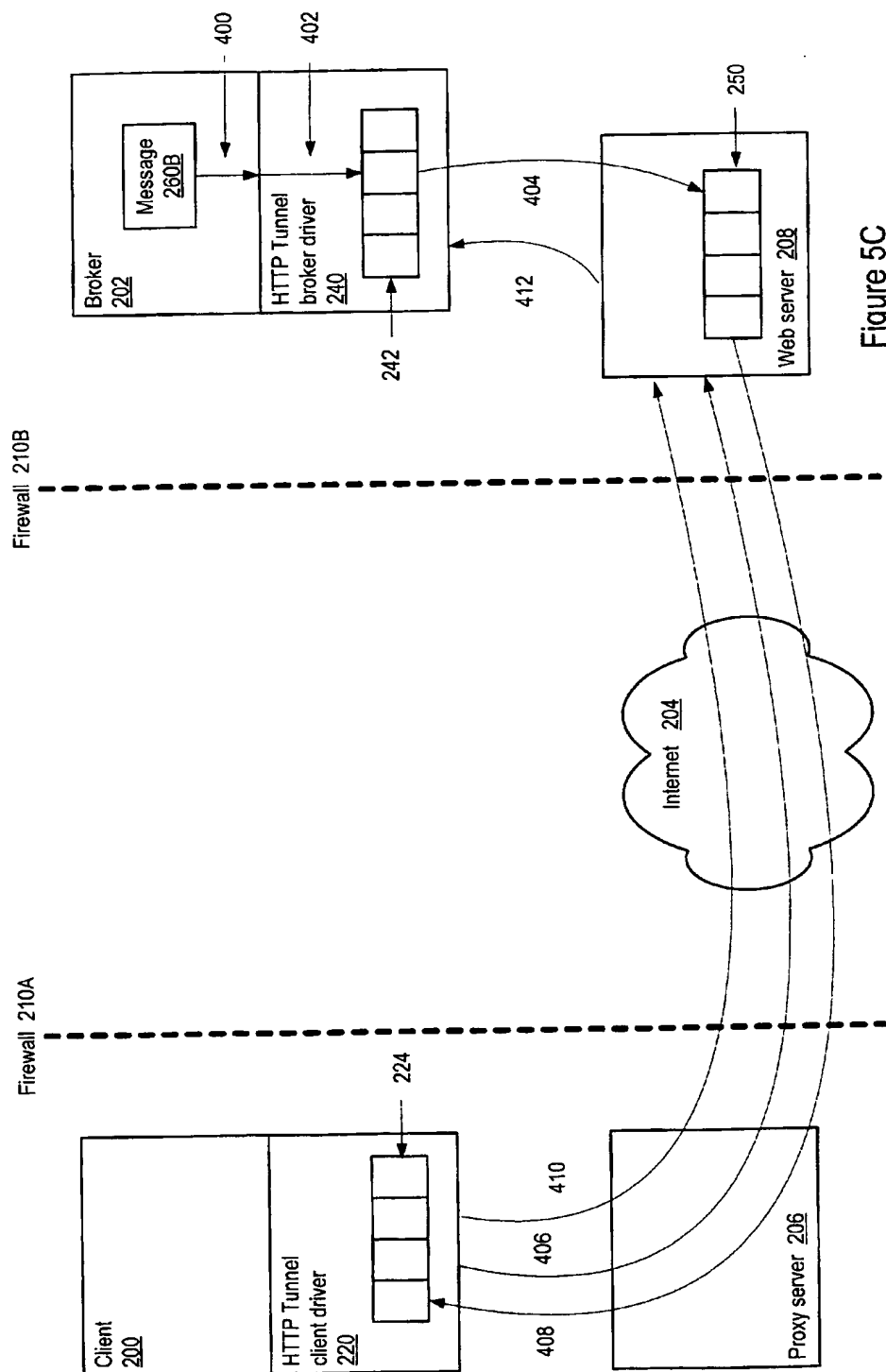


Figure 5C

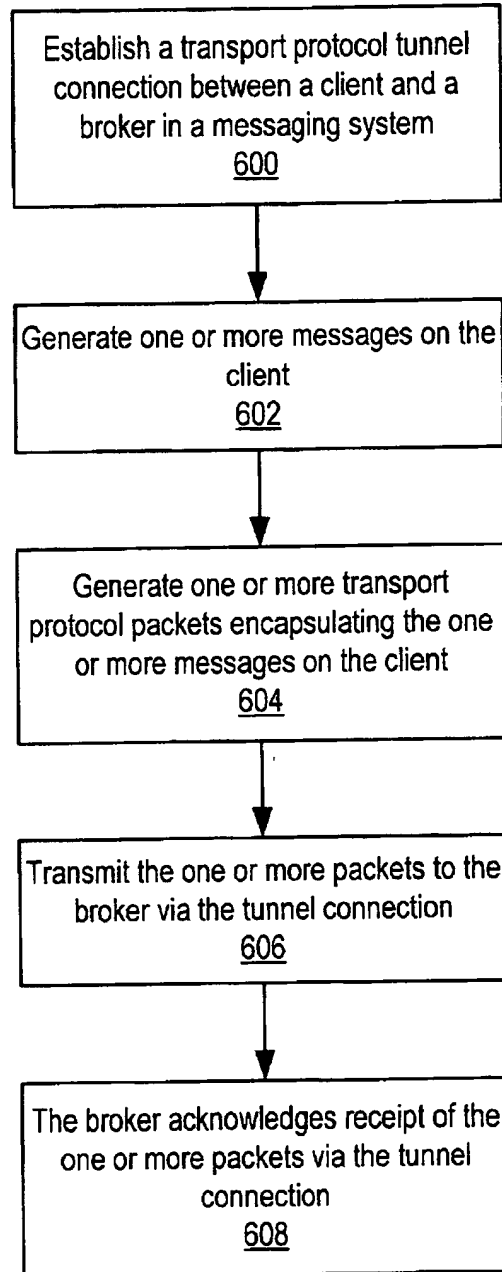


Figure 6A

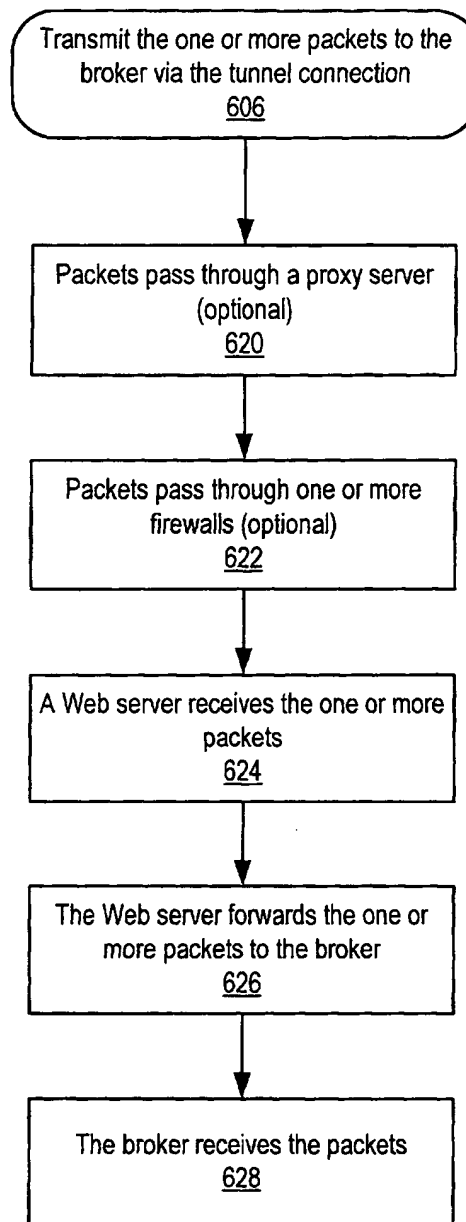


Figure 6B

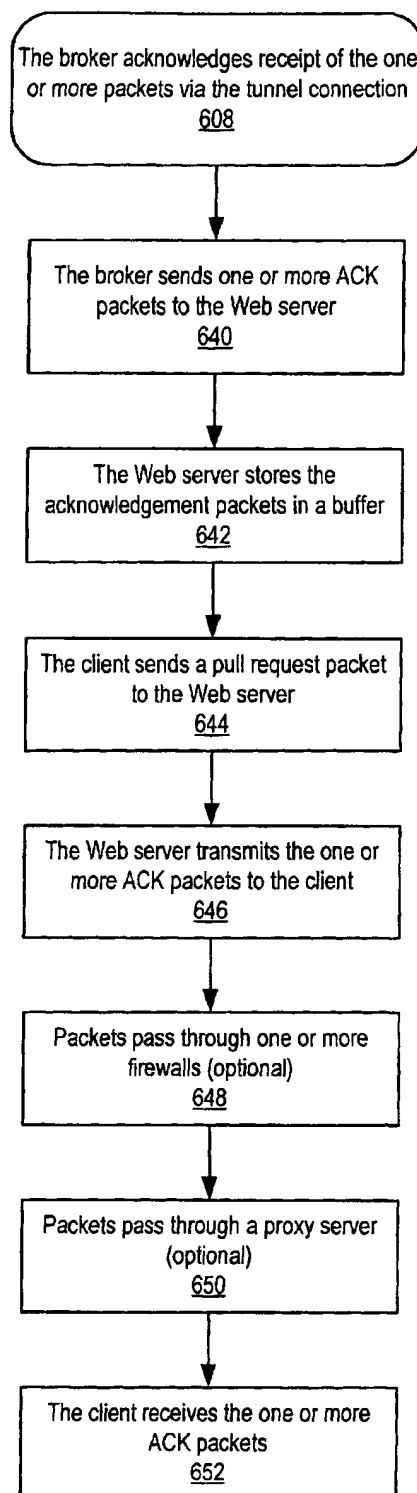


Figure 6C

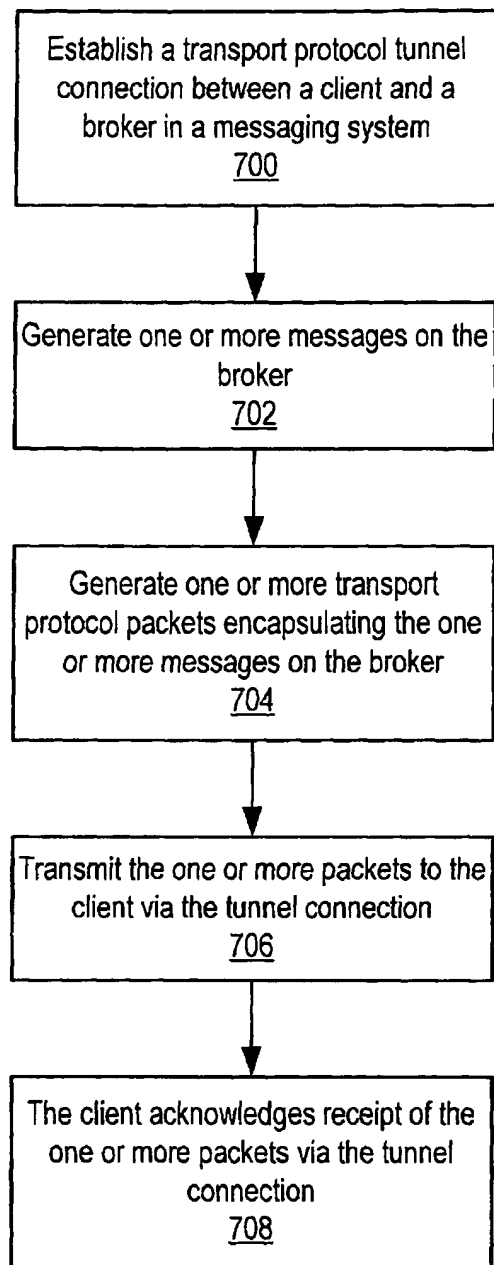


Figure 7A

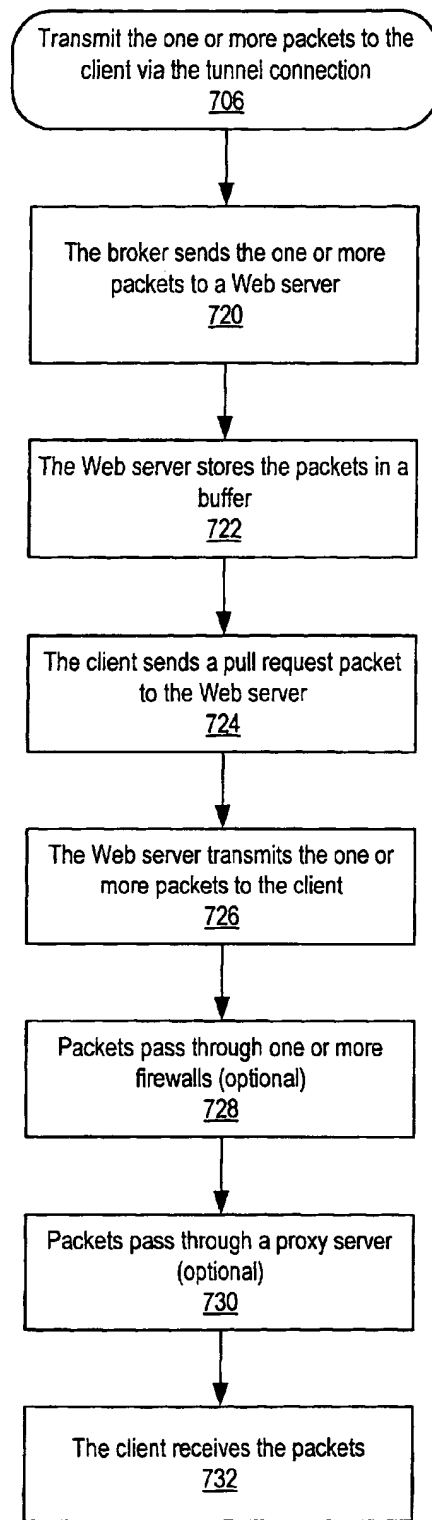


Figure 7B

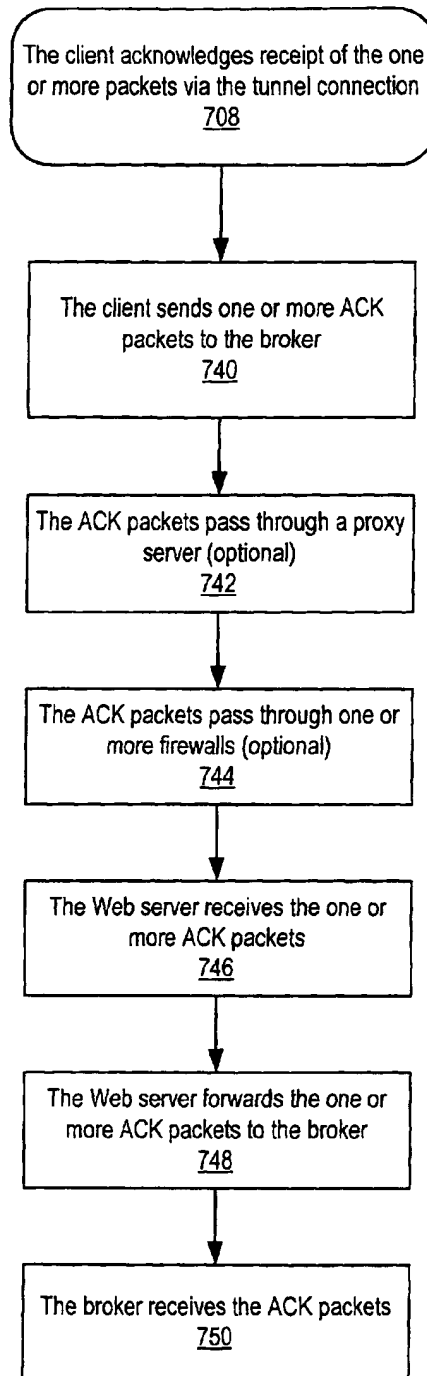


Figure 7C

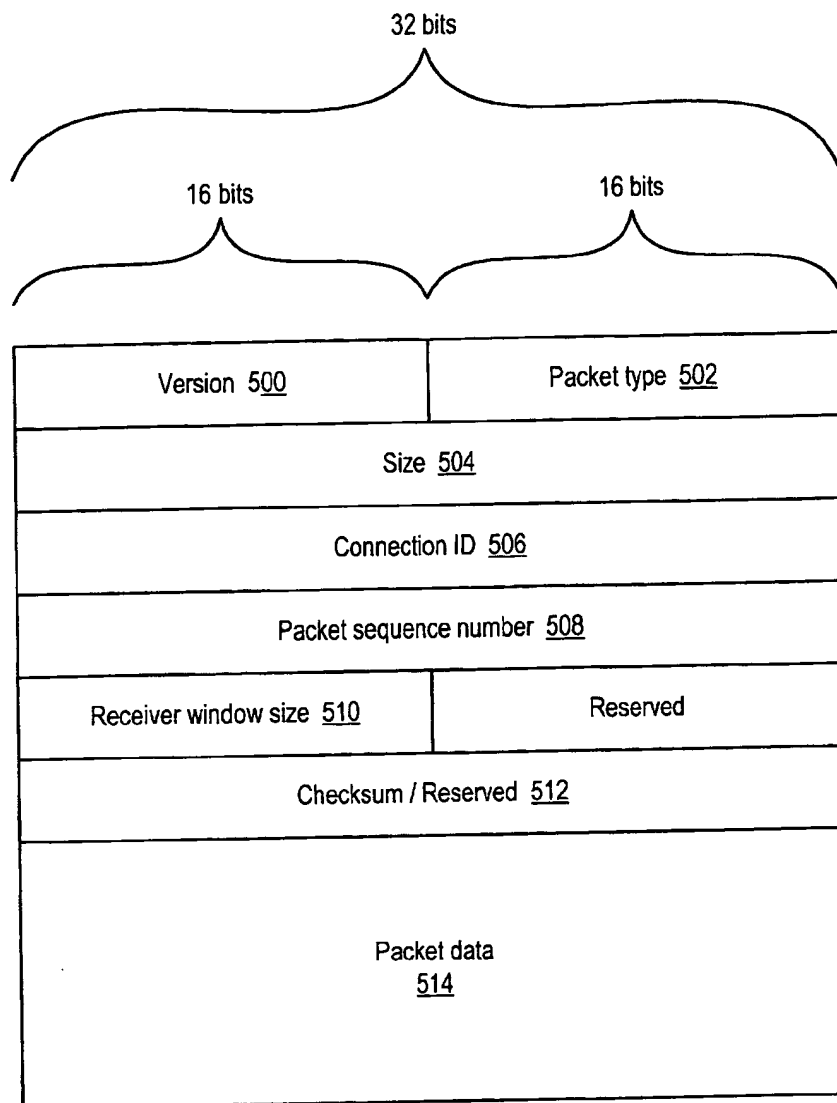


Figure 8

Exemplary tunneling packet format

SYSTEM AND METHOD FOR PROVIDING TUNNEL CONNECTIONS BETWEEN ENTITIES IN A MESSAGING SYSTEM

BACKGROUND OF THE INVENTION

[0001] 1. Field of the Invention

[0002] This invention relates to computers and networks of computers, and more particularly to a system and method for providing transport protocol tunnel connections between entities or nodes such as clients and servers in a messaging system.

[0003] 2. Description of the Related Art

[0004] Messaging is playing an increasingly important role in computing. Its advantages are a natural result of several factors: the trend toward peer-to-peer computing, greater platform heterogeneity, and greater modularity, coupled with the trend away from synchronous communication between processes. The common building block of a messaging service is the message. Messages are specially formatted data describing events, requests, and replies that are created by and delivered to computer programs. Messages contain formatted data with specific meanings. Messaging is the exchange of messages to a messaging server, which acts as a message exchange program for client programs. A messaging server is a middleware program that handles messages that are sent by client programs for use by other programs. Typically, client programs access the functionality of the messaging system using a messaging application program interface (application program interface). A messaging server can usually queue and prioritize messages as needed, and thus saves each of the client programs from having to perform these services. Rather than communicate directly with each other, the components in an application based around a message service send messages to a message server. The message server, in turn, delivers the messages to the specified recipients

[0005] There are two major messaging system models: the point-to-point model and the publish and subscribe model. Messaging allows programs to share common message-handling code, to isolate resources and interdependencies, and to easily handle an increase in message volume. Messaging also makes it easier for programs to communicate across different programming environments (languages, compilers, and operating systems) since the only thing that each environment needs to understand is the common messaging format and protocol. The messages involved exchange crucial data between computers—rather than between users—and contain information such as event notification and service requests. IBM's MQSeries and iPlanet Message Queue are examples of products that provide messaging interfaces and services.

[0006] FIG. 1 illustrates a typical messaging-based application. This application is a modification of the traditional client/server architecture. The major difference is the presence of a messaging server 100A between client 102 and server 104 layers. Thus, rather than communicating directly, clients 102 and servers 104 communicate via the messaging server 100A. The addition of the messaging server 100A adds another layer to the application, but it greatly simplifies the design of both the clients 102 and the servers 104 (they are no longer responsible for handling communications

issues), and it also enhances scalability. Note that servers in a messaging system may also be referred to as "brokers".

[0007] FIG. 2 illustrates another messaging-based application based on point-to-point architecture. This type of application almost demands a centralized messaging server 100B. Without one, each component 106 would be responsible for creating and maintaining connections with the other components 106. A possible alternative approach would be to architect the system around a communication bus, but this would still leave each component 106 in charge of message delivery issues.

[0008] Java Message Service (JMS)

[0009] Java Message Service (JMS) is an application program interface (API) from Sun Microsystems that supports messaging between computers in a network. JMS provides a common interface to standard messaging protocols and also to special messaging services in support of Java programs. Sun advocates the use of the JMS for anyone developing Java applications, which can be run from any major operating system platform. Using the JMS interface, a programmer can invoke the messaging services of IBM's MQSeries, Progress Software's SonicMQ, and other messaging product vendors.

[0010] The JMS API may:

[0011] Provide a single, unified message API

[0012] Provide an API suitable for the creation of messages that match the format used by existing, non-JMS applications

[0013] Support the development of heterogeneous applications that span operating systems, platforms, architectures, and computer languages

[0014] Support messages that contain serialized Java objects

[0015] Support messages that contain eXtensible Markup Language (XML) pages

[0016] Allow messages to be prioritized

[0017] Deliver messages either synchronously or asynchronously

[0018] Guarantee messages are delivered once and only once

[0019] Support message delivery notification

[0020] Support message time-to-live

[0021] Support transactions

[0022] The JMS API is divided into two nearly identical pieces. One implements a point-to-point model of messaging, and the other implements a publish and subscribe model of messaging. Each of these models is called a domain. The APIs are almost identical between the domains. The separation of the API into two domains relieves vendors that support only one messaging model from providing facilities their product doesn't natively support.

[0023] Enterprise Messaging Systems

[0024] Enterprise messaging systems may be developed using a messaging service such as JMS. An enterprise messaging system may be used to integrate distributed,

loosely coupled applications/systems in a way that provides for dynamic topologies of cooperating systems/services. Enterprise messaging systems typically need to address common messaging related problems such as:

[0025] Guaranteed message delivery (e.g. persistence, durable interests, "at least once" and "once and only once" message delivery guarantees, transactions etc). Messages from one component to another must not be lost due to network or system failure. This means the system must be able to guarantee that a message is successfully delivered.

[0026] Asynchronous delivery. For large numbers of components to be able to exchange messages simultaneously, and support high density throughputs, the sending of a message cannot depend upon the readiness of the consumer to immediately receive it. If a consumer is busy or offline, the system must allow for a message to be sent and subsequently received when the consumer is ready. This is known as asynchronous message delivery, popularly known as store-and-forward messaging.

[0027] Various message delivery models (e.g. publish and subscribe or point-to-point).

[0028] Transport independence.

[0029] Leveraging an enterprise messaging system in developing business solutions allows developers to focus on their application/business logic rather than on implementing the underlying messaging layer.

[0030] iPlanet E-Commerce Solutions' iMQ (iplanet Message Queue), formerly offered by Sun Microsystems as JMQ (Java Message Queue) is an example of an enterprise messaging system, and was developed to be JMS-compliant. iMQ may use a "hub and spoke" architecture. Clients use an iMQ client library to exchange messages with an iMQ message server (also referred to as a "broker").

[0031] In an enterprise messaging system, clients exchange messages with a messaging server using a message exchange protocol. The messaging server then may route the messages based upon properties of the messages. Typically, the message exchange protocol requires a direct, fully bi-directional reliable transport connection between the client and the messaging server, such as a TCP (Transport Control Protocol) or SSL (Secure Sockets Layer) connection, which can be used only if the client and the messaging server both reside on the "intranet" (i.e. on the same side of a firewall).

[0032] Hypertext Transfer Protocol

[0033] The Hypertext Transfer Protocol (HTTP) is a set of rules for exchanging files (text, graphic images, sound, video, and other multimedia files) on the World Wide Web. HTTP may also be used on an intranet. Relative to the TCP/IP suite of protocols (which are the basis for information exchange on the Internet), HTTP is an application protocol that is implemented over TCP/IP. HTTP was designed as a stateless request-response mechanism.

[0034] A Web server is a program that, using the client/server model and HTTP, serves the files that form Web pages to Web users (whose computers contain HTTP clients that forward their requests). Every computer on the Internet that

contains a Web site must have a Web server. A Web server machine may include, in addition to the Hypertext Markup Language (HTML) and other files it can serve, an HTTP daemon, a program that is designed to wait for HTTP requests and handle them when they arrive. A Web browser is an example of an HTTP client that sends requests to server machines. When a browser user enters file requests by either "opening" a Web file by typing in a URL (Uniform Resource Locator) or clicking on a hypertext link, the browser builds an HTTP request and sends it to the Internet Protocol (IP) address indicated by the URL. The HTTP daemon in the destination server machine receives the request and, after any necessary processing, the requested file is returned.

[0035] Tunneling

[0036] Tunneling may be defined as the encapsulation of a protocol A within protocol B, such that A treats B as though it were a data link layer. Tunneling may be used to get data between administrative domains that use a protocol that is not supported by the Internet connecting those domains. A "tunnel" is a particular path that a given message or file might travel through the Internet.

[0037] Proxy Servers and Firewalls.

[0038] In an enterprise that uses the Internet, a proxy server is a server that acts as an intermediary between a workstation user and the Internet so that the enterprise can ensure security, administrative control, and caching service. A proxy server may be associated with or part of a gateway server that separates the enterprise network from the outside network and a firewall server that protects the enterprise network from outside intrusion.

[0039] A firewall is a set of related programs, usually located at a network gateway server, that protects the resources of a private network from users from other networks. An enterprise with an intranet that allows its workers access to the wider Internet installs a firewall to prevent outsiders from accessing its own private data resources and for controlling what outside resources its own users have access to.

SUMMARY OF THE INVENTION

[0040] A system and method for providing transport protocol tunnel connections between entities such as clients and servers in a messaging system is described. A transport protocol tunnel connection layer is described that may be used to provide reliable, full duplex virtual connections between entities (e.g. clients and servers) in a distributed application environment using a messaging system. This layer may be used by clients to access messaging servers (referred to as brokers) through proxy servers and firewalls, thus expanding the scope of from where clients can access brokers. This layer allows for information flow in both directions between the client and the server. Using this layer, brokers as well as clients may initiate messages in the distributed application environment. This layer may also provide guaranteed data delivery with correct sequencing even in case of a failure on the network. This layer may also provide end-to-end flow control.

[0041] The transport protocol tunnel connection layer may provide a connection-oriented, bi-directional byte stream service between nodes in a messaging system. Application data, including messages, may be carried as transport pro-

protocol packet payloads. The transport protocol tunnel connection layer allows messaging system clients to access messaging system brokers using the transport protocol instead of using direct TCP connections. This enables the clients to access the brokers through firewalls. Also, with the help of a transport protocol proxy, the clients may access the messaging service even when there is no direct IP connectivity with the broker.

[0042] Using embodiments of the transport protocol tunnel connection layer, a transport protocol tunnel connection between the client and the broker in the messaging system may be established. The client may then generate one or more messaging system messages. In one embodiment, the generated messages may then be stored in a client-side transmit buffer. One or more transport protocol packets encapsulating the one or more messages may then be generated on the client. In one embodiment, a client-side tunnel connection driver may generate the packets and include the messages as payloads of the packets. The one or more transport protocol packets may then be transmitted to the broker via the tunnel connection. In one embodiment, the client-side tunnel connection driver may handle the transmission of the packets.

[0043] A Web server may then receive the one or more transport protocol packets. The Web server may then forward the received transport protocol packets to the broker. In one embodiment, the packets may be forwarded to the broker via a TCP connection serving as one segment of the transport protocol tunnel connection between the Web server and the broker. In one embodiment, a transport protocol tunnel servlet on the Web server node may serve as an interface between the Web server and the TCP connection.

[0044] The broker may receive the transport protocol packets from the Web server. In one embodiment, a broker-side transport protocol tunnel driver may receive the packets. In one embodiment, the broker may extract the messaging system messages from the transport protocol packets and store the messages in a broker-side receive buffer. In another embodiment, the entire transport protocol packet may be stored in the broker-side receive buffer.

[0045] The broker may acknowledge receipt of the packets by sending one or more acknowledgement (ACK) packets to the client via the tunnel connection. The broker may generate and send one or more acknowledgement (ACK) packets to the Web server. In one embodiment, the broker may store ACK packets in a broker-side transmit buffer. In one embodiment, the ACK packets may be sent to the Web server over the TCP connection. In one embodiment, the broker-side transport protocol tunnel driver may handle the transmission of the ACK packets to the Web server.

[0046] The Web server may then receive the ACK packets. In one embodiment, a transport protocol tunnel servlet may receive the packets for the Web server. The Web server may store the acknowledgement packets in a transport protocol packet buffer. At some point, the client may send a transport protocol pull request packet to the Web server. In one embodiment, the client may periodically send pull requests to the Web server. In one embodiment, a separate thread on the client may handle periodically sending pull requests. The Web server may transmit to the client the one or more transport protocol ACK packets stored in the transport protocol packet buffer associated with the client in response

to receiving the pull request packet. The client may then receive the one or more ACK packets. The ACK packets may serve to acknowledge the receipt of the transmitted data so that the sender may free its transmit buffers.

[0047] Using embodiments of the transport protocol tunnel connection layer, messaging system messages may be generated on a broker and sent to a client. In one embodiment, the generated messages may be stored in a broker-side transmit buffer. One or more transport protocol packets encapsulating the one or more messages may then be generated on the broker. In one embodiment, a broker-side transport protocol tunnel connection driver may generate the transport protocol packets and include the messages as payloads of the packets. The one or more transport protocol packets may then be transmitted to the client via the transport protocol tunnel connection. In one embodiment, the broker-side transport protocol tunnel connection driver may handle the transmission of the packets. In one embodiment, the transport protocol packets may be sent to a Web server. In one embodiment, the transport protocol packets may be sent to the Web server over a TCP connection.

[0048] The Web server may receive the transport protocol packets and store the received packets in a transport protocol packet buffer. In one embodiment, a transport protocol tunnel servlet may receive the packets for the Web server. At some point, the client may send a transport protocol pull request packet to the Web server. The Web server may transmit to the client one or more transport protocol packets stored in the transport protocol packet buffer associated with the client in response to receiving the pull request packet.

[0049] The client may then receive the one or more transport protocol packets. In one embodiment, the client may store the received transport protocol packets in a client-side receive buffer. After receiving the transport protocol packets, the client may acknowledge receipt of the packets by sending one or more acknowledgement (ACK) packets to the broker via the transport protocol tunnel connection. The Web server may receive the ACK packets and forward the received ACK packets to the broker. In one embodiment, the ACK packets may be transmitted to the broker via a TCP connection between the Web server and the broker that serves as one segment of the HTTP tunnel connection. The broker may receive the ACK packets from the Web server. In one embodiment, the received ACK packets may be stored in a broker-side receive buffer.

[0050] In one embodiment, one Web server and one tunnel servlet may be used by two or more clients to communicate to a broker via tunnel connections. In this embodiment, the HTTP tunnel servlet may multiplex transport protocol packets from the two or more clients onto the TCP connection. In one embodiment, the tunnel servlet may extract the messaging system message information from received transport protocol packets and send only the message information to the broker over the TCP connection. In one embodiment, there may be one broker-side receive buffer for each tunnel connection. In another embodiment, a single receive buffer may be used for two or more tunnel connections.

[0051] Transport protocol packets may optionally pass through a proxy server and one or more firewalls. For example, transport protocol packet flow from a client to a broker may pass through a proxy server, through a firewall onto the Internet, through another firewall to a Web server,

and from the Web server over a TCP connection to the broker. Packets flowing in the opposite direction (from the broker to the client) take the reverse path.

[0052] In one embodiment, transport protocol packets transmitted on the tunnel connection may include message sequence information configured for use by the receiver in processing received messages in the correct sequence. In one embodiment, flow control may be applied to the sending of messages from a sender to a receiver. In one embodiment, the receiver may inform the sender of available space in a receive buffer to store incoming messages. In one embodiment, upon establishment of a tunnel connection, each side (both clients and brokers can be senders and/or receivers) may inform the other of its receive buffer size. In one embodiment, ACK packets sent to a sender by a receiver may each include the currently available space in the receive buffer. Thus, the sender can keep track of the current receive capacity of the receiver.

BRIEF DESCRIPTION OF THE DRAWINGS

[0053] FIG. 1 illustrates a prior art messaging-based application based upon client/server architecture;

[0054] FIG. 2 illustrates a prior art messaging-based application based on point-to-point architecture;

[0055] FIG. 3A illustrates a client-server messaging system implementing a transport protocol tunnel connection layer according to one embodiment;

[0056] FIG. 3B illustrates a client-server messaging system implementing a transport protocol tunnel connection layer with multiple clients accessing a broker through a Web server according to one embodiment;

[0057] FIG. 4 illustrates the architecture of a client-server messaging system implementing a transport protocol tunnel connection layer according to one embodiment;

[0058] FIG. 5A illustrates the routing of transport protocol packets between clients and a broker on transport protocol tunnel connections according to one embodiment;

[0059] FIG. 5B illustrates the process of sending a message from a client to a broker via a transport protocol tunnel connection according to one embodiment;

[0060] FIG. 5C illustrates the process of sending a message from a broker to a client via a transport protocol tunnel connection according to one embodiment;

[0061] FIGS. 6A is a flowchart of a method for sending messages from a messaging system client to a messaging system broker over a transport protocol tunnel connection layer according to one embodiment; FIG. 6B is a flowchart of a method for transmitting one or more packets from a client to a broker via a transport protocol tunnel connection according to one embodiment;

[0062] FIG. 6C is a flowchart of a method for a broker to acknowledge to a client the receipt of transport protocol packets via the transport protocol tunnel connection according to one embodiment;

[0063] FIG. 7A is a flowchart of a method for sending messages from a messaging system client to a messaging system broker over a transport protocol tunnel connection layer according to one embodiment;

[0064] FIG. 7B is a flowchart of a method for transmitting one or more packets from a broker to a client via a transport protocol tunnel connection according to one embodiment;

[0065] FIG. 7C is a flowchart of a method for a client to acknowledge to a broker the receipt of transport protocol packets via a transport protocol tunnel connection according to one embodiment; and

[0066] FIG. 8 illustrates an exemplary transport protocol packet format that may be used in the transport protocol tunnel connection layer according to one embodiment.

[0067] While the invention is described herein by way of example for several embodiments and illustrative drawings, those skilled in the art will recognize that the invention is not limited to the embodiments or drawings described. It should be understood, that the drawings and detailed description thereto are not intended to limit the invention to the particular form disclosed, but on the contrary, the intention is to cover all modifications, equivalents and alternatives falling within the spirit and scope of the present invention as defined by the appended claims. The headings used herein are for organizational purposes only and are not meant to be used to limit the scope of the description or the claims. As used throughout this application, the word "may" is used in a permissive sense (i.e., meaning having the potential to), rather than the mandatory sense (i.e., meaning must). Similarly, the words "include", "including", and "includes" mean including, but not limited to.

DETAILED DESCRIPTION OF EMBODIMENTS OF THE INVENTION

[0068] A system and method for providing transport protocol tunnel connections between entities such as clients and servers in a messaging system is described. A transport protocol tunnel connection layer is described that may be used to provide reliable, full duplex virtual connections between entities (e.g. clients and servers) in a distributed application environment using a messaging system. This layer may be used by clients to access messaging servers (referred to as brokers) through proxy servers and firewalls, thus expanding the scope of from where clients can access brokers. Using this layer, brokers as well as clients may initiate messages in the distributed application environment. This layer may also provide guaranteed data delivery with correct sequencing even in case of a failure on the network. This layer may also provide end-to-end flow control. The transport protocol tunnel connection layer may be used to simulate virtual connections that have a similar contract with the upper layers (e.g. clients and brokers) as a TCP connection. Thus, applications can be developed in terms of messages without concern for the particular underlying connection protocol. Decisions about the best communications protocol between clients and brokers may be postponed until deployment time when the particular network requirements of an installation are known.

[0069] The transport protocol tunnel connection layer may use a novel tunneling protocol to overcome limitations of the underlying transport protocol in the messaging environment. A transport protocol may be designed as a stateless request-response mechanism, and thus may not fit the enterprise messaging system protocol model very well. For example, enterprise messaging system applications may expect asynchronous message delivery whereas a transport protocol

may use a synchronous request response model. As another example, each transport protocol request-response exchange may carry a finite amount of data and is usually short lived. Thus, an enterprise messaging system message may be delivered using multiple transport protocol requests. However there may be no correlation between transport protocol requests generated by the same client. The Web servers and transport protocol intermediaries (e.g. proxy servers) thus cannot guarantee that the transport protocol requests will be processed in the same sequence as they were sent. As yet another example, TCP protocol uses a flow control mechanism to limit the network resource usage. If an enterprise messaging system message sender application generates messages very rapidly, and if each message is sent to the server using a separate transport protocol request or multiple transport protocol requests, resources on the transport protocol servers and intermediaries may be exhausted. As still yet another example, an enterprise messaging system client application may maintain a connection and a steady message exchange rate over very long periods of time (many days or even months). If the transport protocol is used, a failure on a common transport protocol proxy server may disrupt the communication between the client application and the enterprise messaging system broker.

[0070] The transport protocol tunnel connection layer may provide a connection-oriented, bi-directional byte stream service between nodes in a messaging system. Transport protocol messages may be carried as transport protocol packet payloads. The transport protocol tunnel connection layer may allow messaging system clients to access messaging system brokers using the transport protocol instead of using direct TCP connections. This enables the clients to access the brokers through firewalls. Also, with the help of a transport protocol proxy, the clients may access the messaging service even when there is no direct IP connectivity with the broker.

[0071] Using the transport protocol tunnel connection layer, if a client is separated from a broker by a firewall, messaging may be run on top of transport protocol connections, which are normally allowed through firewalls. On the client side, a transport protocol transport driver may encapsulate messages into transport protocol packets, and also may ensure that these packets are sent to the Web server in the correct sequence. The client may use a transport protocol proxy server to communicate with the broker if necessary. In one embodiment, a transport protocol tunnel servlet executing within a Web server may receive the transport protocol packets and forward the entire transport protocol packets (including the message data) to the broker. In another embodiment, the transport protocol tunnel servlet executing within the Web server may be used to pull client data (messages) out of the transport protocol packets before forwarding the data to the broker. In one embodiment, the tunnel servlet may multiplex message data from multiple clients onto one TCP connection to the broker, thus allowing the use of one tunnel servlet for multiple clients. The transport protocol tunnel servlet may also send broker data (messages) to the client in response to transport protocol pull requests. On the broker side, a transport protocol transport driver may unwrap and demultiplex incoming messages from the transport protocol tunnel servlet. The transport protocol tunnel connection layer may also be used by brokers to initiate communications with a client.

[0072] Though embodiments of the system and method are described herein as providing Hypertext Transport Protocol (HTTP) tunnel connections between entities such as clients and servers in a messaging system, it is noted that embodiments of the system and method using other unreliable/connectionless transport protocols that may not have built-in TCP support such as UDP (User Datagram Protocol), IrDA (Infrared Data Association), IBM's SNA (Systems Network Architecture), Novell's IPX (Internetwork Packet eXchange), and Bluetooth are contemplated. These embodiments may be used to provide tunnel connections between entities in messaging systems using the other transport protocols.

[0073] FIG. 3A illustrates a client-server messaging system implementing an HTTP tunnel connection layer according to one embodiment. Client 200 may generate messages using the messaging protocol 212. In one embodiment, an Application Programming Interface (API) to the messaging protocol may be used by the client application to generate the messages. In one embodiment, the messaging protocol is the Java Message Service (JMS). Other messaging protocols may be used. Generated messages may then be passed to the HTTP tunnel client driver 220.

[0074] The HTTP tunnel client driver 220 may then send the messages as HTTP POST request payloads. The HTTP tunnel client driver 220 may also use separate HTTP requests to periodically pull any data sent by the other end of the connection. The HTTP requests may be sent through HTTP proxy 206, Internet 204, and firewall 210 to Web server 208. On Web server 208, the HTTP tunnel servlet 214 may act as a transceiver and may multiplex the HTTP requests from multiple clients onto a single TCP connection 216 with the broker 202. The HTTP tunnel broker driver 240 may receive the HTTP requests from the Web server 210 over TCP connection 216.

[0075] Note that the HTTP proxy 206 and/or the firewall 210 are optional. In other words, the HTTP tunnel connection layer is configurable to transmit messages encapsulated in HTTP requests between entities over the Internet 104 both with and without the messages passing through proxies and/or firewalls. Also note that the Web server 208, HTTP tunnel servlet 214, and the broker 202 may be implemented on the same host machine or on different host machines. Note also that a Web server 208 and HTTP tunnel servlet 214 may be used to access multiple brokers 202.

[0076] In one embodiment, the packet delivery service provided by the HTTP tunnel drivers may occasionally lose packets. Hence the HTTP tunnel connection layer may use positive acknowledgements of received packets, and may retransmit any lost packets. When a receiver receives a packet including a message, the receiver responds to the packet by sending an acknowledgement packet to the sender.

[0077] In one embodiment, the HTTP tunnel connection layer may use a sliding window protocol to implement packet sequencing and flow control on top of HTTP. In a distributed application environment using a messaging service, quite often entities may need to send a stream of packets, without errors, and with guaranteed order of delivery (i.e. that the packets are received in the order they are sent). HTTP does not guarantee packets to be received by a receiver (e.g. broker) in the same order the packets were sent by a transmitter (e.g. client). Also, HTTP does not provide

a method to control the rate at which packets are sent to a receiver, thus running the risk of exhausting network resources on the receiver and losing packets. Using a sliding window protocol provides the ability to control the rate at which packets are transmitted to a receiver. The sliding window protocol may be used to guarantee that no more than a fixed number of packets are transmitted to a receiver. The fixed number of packets may be determined by a receive buffer size on the receiver. For example, a receiver may be able to receive a maximum of 100 packets from a sender. If the sender initially transmits 70 packets, the receiver may receive the packets and process 20 of them. The receiver may send a packet or packets to notify the sender that the receiver can receive 50 packets. The sender may then transmit 30 more packets. The receiver may receive the 30 packets, and in the meantime may have processed 20 more of the original 70 packets. The receiver may then transmit a packet or packets to notify the sender that the receiver can receive 60 packets (there are still 10 of the original 70 packets and the 30 new packets in the receive buffer). Thus, the sender never sends more packets to the receiver than the quantity of packets the receiver has notified the sender it can receive.

[0078] In one embodiment, when a connection is established between two entities or nodes (e.g. a server and client), each node may inform the other of how many packets it can initially receive. In a network, a node is a connection point, either a redistribution point or an end point for data transmissions. In general, a node has programmed or engineered capability to recognize, process and/or forward transmissions to other nodes. In one embodiment, each packet received by the receiver may be acknowledged with a packet sent to the sender. The acknowledgement packets may each include information indicating the current receive buffer size (i.e. the number of packets that the receiver can currently receive). Thus, the sender can determine the number of packets that it can send to the receiver without overwhelming the sender.

[0079] In one embodiment, the client 200, broker 202 and the messaging protocol layers 212 may function similarly whether the underlying transport protocol is TCP or HTTP. In one embodiment, the messaging protocol layers 212 on both the client and the broker may use the same basic design and threading model for TCP and HTTP support. In one embodiment, an enterprise messaging system using the HTTP tunneling protocol may allow a messaging system application to exchange messages using the TCP, HTTP, SSL, or other protocol by changing appropriate configuration parameters at runtime. Thus, the application developer may not have to write any transport specific code. A client library and the broker 202 may handle the transport-specific details.

[0080] FIG. 3B illustrates a client-server messaging system implementing the HTTP tunnel connection layer with multiple clients accessing a broker through a Web server according to one embodiment. Both brokers 202A and 202B may have registered with Web server 208 that they are ready to receive connections from clients. Client 200A may establish an HTTP tunnel connection to broker 202A through Web server 208. Client 200B may establish an HTTP tunnel connection to broker 202B through Web server 208. At some point, either client may establish an HTTP tunnel connection with the other broker. Thus, a client may have multiple

HTTP tunnel connections open to different brokers, and multiple clients may have HTTP tunnel connections open to one broker.

[0081] FIG. 4 illustrates the architecture of a client-server messaging system implementing the HTTP tunnel connection layer according to one embodiment. Various components that are comprised in the network protocol stack for the HTTP tunnel connection layer are shown. These components lie between the application (client or broker) and the HTTP tunnel driver (client or broker). These components may include an HTTP tunnel input stream/HTTP tunnel output stream 270, an HTTP tunnel socket 272, and an HTTP tunnel connection 274. In one embodiment, these components may be implemented as classes. In one embodiment, these components may be implemented as Java classes.

[0082] The HTTP tunnel drivers may send one request or receive one packet at a time. The primary responsibility of these drivers is to make sure that single packets are sent and/or received. This driver may not be aware of or be involved directly in the ordering, flow control or acknowledgement of packets.

[0083] The HTTP tunnel connection component 274 may interpret received packets. This component may be responsible for the sliding window protocol implementation. This component may be responsible for buffering the packets, may implement flow control, reordering and other aspects of the HTTP tunneling protocol. This component may be the primary component that is used to implement the end-to-end HTTP tunnel "virtual" connection as described herein.

[0084] The HTTP tunnel socket 272 and input and output streams 270 provide an interface between the application (e.g. client or broker) and the HTTP tunnel connection 274. These components may include simple, atomic methods such as read, write and open connection methods that provide an abstract interface to the HTTP tunnel connection 274 for the applications, and thus may hide the HTTP tunnel connection implementation from the applications.

[0085] HTTP tunnel server socket 276 may be used by broker 202 to open a listening socket to the Web server 208 to allow the Web server to create a listening endpoint for the broker 202. In doing so, broker 202 is establishing that it is ready to accept HTTP tunnel connections through Web server 208. Thus, when client 200 desires to open an HTTP tunnel connection to broker 202, the client may contact Web server 208 with a packet including information identifying broker 202. Web server 208 may examine the packet and forward the packet to broker 202, where an HTTP tunnel connection to client 200 may be established. Note that one or more HTTP packet buffers may be allocated on Web server 208 for the newly established HTTP tunnel connection. Thus, a client 200 initiates a connection, and a broker 202 accepts the connection. In one embodiment, brokers 202 cannot initiate connections to clients. Therefore, once an HTTP tunnel connection between a client 200 and a broker 202 is open (initiated by the client 200), the client 200 may keep the connection open so that if the broker 202 has data to send to the client, there is a communications link established for the broker to send messages to the client on. In one embodiment, the broker 202 never sends HTTP packets directly to the client 200; the packets are buffered on Web server 208, and pulled from the Web server periodically by the client 200.

[0086] FIGS. 5A through 5C are data flow diagrams illustrating the operation of a client-broker messaging system implementing the HTTP tunnel connection layer according to one embodiment. FIG. 5A illustrates the routing of HTTP packets between clients 200 and a broker 202 on HTTP tunnel connections 292 according to one embodiment. Each client 200 may include at least one client-side transmit buffer 222 and at least one client-side receive buffer 224. Broker 202 may include at least one broker-side receive buffer 244 and at least one broker-side transmit buffer 242. Each client 200 may establish an HTTP tunnel connection 292 to broker 202, which may pass through network 294, through Web server 208, and over a TCP connection 296 to broker 202. There may be a single TCP connection 296 between Web server 208, or alternatively a TCP connection 296 may be established for each HTTP tunnel connection 292. In passing through network 294, an HTTP tunnel connection 292 may pass through a proxy server (not shown) and/or through one or more firewalls (not shown).

[0087] Clients 200 may generate messaging system messages, which may be buffered in a client-side transmit buffer 222. Buffering message data may allow messages to be retransmitted if necessary, for example. The clients 200 may generate HTTP packets that include the message data as payloads and transmit the HTTP message packets to broker 202 via the HTTP tunnel connections 292 as indicated at 280A and 280B. Web server 208 may receive the HTTP message packets from network 294 and forward the packets to broker 202 over a TCP connection 296.

[0088] Broker 202 may buffer incoming messages in a broker-side receive buffer 244. Broker 202 may generate an acknowledgment (ACK) HTTP packet to acknowledge the receipt of each HTTP packet successfully received from a client 200, and send the ACK packets to Web server 208 over a TCP connection 296 as indicated at 282A and 282B. Web server 208 may buffer the ACK packets received from broker 202 in buffers 250. In one embodiment, there may be a buffer 250 for each HTTP tunnel connection 292; in other words, each connection 292 may use a separate instance of buffer 250. In another embodiment, one buffer 250 may be shared among two or more HTTP tunnel connections 292.

[0089] Using the HTTP tunneling protocol layer, a broker 202 as well as clients 200 may initiate messaging system messages. Messages generated by broker 202 may be buffered in a broker-side transmit buffer 242. Buffering message data may allow messages to be retransmitted if necessary, for example. The broker 202 may generate HTTP packets that include the message data as payloads and transmit the HTTP packets to Web server 208 over a TCP connection 296 as indicated at 288A and 288B. Web server 208 may buffer the HTTP message packets received from broker 202 in buffers 250.

[0090] As indicated at 284A and 284B, each client 200 may send an HTTP request packet to Web server 208 to indicate that the client 200 is ready to receive HTTP packets buffered in a buffer 250 for the client 200. In one embodiment, each client may periodically send HTTP request packets to retrieve buffered HTTP packets from Web server 208. In one embodiment, a separate thread on a client 200 may be responsible for periodically sending the HTTP request packets.

[0091] After Web server 208 receives an HTTP request packet from a client 200 as indicated at 284, the Web server

208 may respond by sending the requesting client 200 one or more HTTP packets currently buffered in a buffer 250 for the client 200. The HTTP packets may include ACK packets as sent at 282 and/or HTTP message packets as sent at 288. Upon receiving HTTP packets from the Web server 208, a client 200 may store the received packets in a client-side receive buffer 224 to await processing. A client 200 may also generate an ACK packet and send them to broker 202 over the HTTP tunnel connection, as indicated at 290A and 290B, in response to each HTTP message packet received.

[0092] FIG. 5B illustrates the process of sending a message from a client 200 to a broker 202 via an HTTP tunnel connection according to one embodiment. At 300, the client 200 may generate a message 260A. At 302, the client side HTTP tunneling driver 220 may receive the message 260A and buffer the outgoing message data in a transmit buffer 222, which may allow the message data to be retransmitted if necessary. The client side HTTP tunneling driver 220 generates an HTTP POST request with the message data as payload as indicated at 304. This HTTP request may travel over the Internet. The client 200 may be configured to send HTTP requests via a proxy server 206 and through one or more firewalls 210 if necessary.

[0093] A Web server 208 may receive the HTTP request with the message data, and forward the HTTP request to the server side HTTP tunneling driver 240 over a TCP connection as indicated at 306. The server side HTTP tunneling driver 240 may store the received message data in a receive buffer 244. The packet header of the HTTP request may include a sequence number for use in preserving message order when multiple messages are transmitted. The message data may remain in receive buffer 244 until the broker 202 consumes it.

[0094] The server side HTTP tunneling driver may generate an acknowledgement (ACK) HTTP packet to indicate successful receipt of the HTTP request packet and message data. The ACK packet may include information about how much space is left in the receive buffer 244. This information may be used as a "flow control" mechanism to slow down the sender (client) if the receiver (broker) cannot consume the data fast enough. The ACK packet may be sent to the Web server 208 over the TCP connection as indicated at 308. The ACK packet may be stored in packet buffer 250A on the Web server 208 waiting for an HTTP request from the client 200. In one embodiment, the Web server 208 may not initiate communication with the client 200; it can only respond to incoming HTTP requests.

[0095] The client side HTTP tunneling driver 220 then may send an HTTP request packet to the Web server 208 to pull any pending HTTP packets as indicated at 310. In one embodiment, the client side HTTP tunneling driver 220 may use a separate reader thread that continuously sends requests to the Web server 208 to pull any pending HTTP packets. After the Web server 208 receives a pull request for the HTTP packet buffered at 308, the Web server may send the buffered HTTP packet(s) to client 200 in response to the pull request as indicated at 312. The client side HTTP tunneling driver 220 then may process the HTTP packet(s) including the ACK packet, and free the corresponding message data buffered in transmit buffer 222 at 302. In one embodiment, information from the HTTP packet(s) sent to the client 200 may be stored in a client-side receive buffer (not shown) and accessed from the client-side receive buffer for processing.

[0096] FIG. 5C illustrates the process of sending a message from the broker 202 to the client 200 via an HTTP tunnel connection according to one embodiment. At 400, the broker 202 generates a message 260B. At 402, the broker side HTTP tunneling driver 240 may receive the message 260B and buffer the outgoing message data in a transmit buffer 242 so that it can be retransmitted if necessary. The broker side HTTP tunneling driver 240 may generate an HTTP packet with the message data as payload and forward it to Web server 208 over a TCP connection as indicated at 404. The Web server 208 receives the HTTP packet with the message data, and writes the packet to buffer 250. The server side HTTP tunneling driver 240 may send an HTTP request to the Web server 208 to pull any pending packets as indicated at 406. When the Web server 208 receives the pull request, it sends the packet buffered at 404 in response to the pull request as indicated at 408. The client side HTTP tunneling driver 220 may store the received message data in a receive buffer 224. The packet header of the HTTP request may include a sequence number to preserve message order. The message data may remain in receive buffer 224 until the client 200 consumes it.

[0097] The client side HTTP tunneling driver generates an acknowledgement (ACK) HTTP packet to indicate successful receipt of the HTTP request and message data. The ACK packet may also include information about how much space is left in receive buffer 224. This information may be used as a "flow control" mechanism to slow down the sender (broker) if the receiver (client) cannot consume the data fast enough. The ACK packet may be sent to the Web server 208 over the Internet as indicated at 410. Web server 208 may forward the ACK packet to the server side HTTP tunneling driver 240 over a TCP connection as indicated at 412. The server side HTTP tunneling driver 240 then may free the corresponding message data packet(s) buffered in transmit buffer 242 at 402. In one embodiment, information from the ACK packet sent to the broker 202 may be stored in a broker-side receive buffer (not shown) and accessed from the broker-side receive buffer for processing.

[0098] FIGS. 6A-6C are flowcharts illustrating a method of sending messages from a messaging system client to a messaging system broker over an HTTP tunnel connection layer according to one embodiment. In FIG. 6A, a transport protocol tunnel connection between a client and a broker in a messaging system may be established as indicated at 600. As indicated at 602, one or more messaging system messages may be generated on the client. In one embodiment, the generated messages may then be stored in a client-side transmit buffer. One or more transport protocol packets encapsulating the one or more messages may be generated on the client as indicated at 604. In one embodiment, a client-side HTTP tunnel connection driver may generate the HTTP packets and include the messages as payloads of the packets. As indicated at 606, the one or more HTTP packets may then be transmitted to the broker via the HTTP tunnel connection. In one embodiment, the client-side HTTP tunnel connection driver may handle the transmission of the packets.

[0099] In one embodiment, each HTTP packet transmitted on the HTTP tunnel connection may include message sequence information configured for use by the receiver in processing received messages in the correct sequence. This is useful since HTTP and some other transport protocols do

not guarantee delivery of messages in order. In one embodiment, flow control may be applied to the sending of messages from a sender to a receiver. In flow control, before sending new messages, the sender may determine available resources on the receiver to receive new messages, and then transmit no more messages than the receiver can handle based upon the available resources. In one embodiment, the receiver may inform the sender of available space in a receive buffer to store incoming messages awaiting processing. In one embodiment, upon establishment of an HTTP tunnel connection, each side (both clients and brokers can be senders and/or receivers) may inform the other of its receive buffer size.

[0100] As indicated at 608, the broker may receive the HTTP packets and then may acknowledge receipt of the one or more packets by sending one or more acknowledgement (ACK) packets to the client via the HTTP tunnel connection. In one embodiment, the ACK packets may each include the currently available space in the broker-side receive buffer. Thus, the sender (client) can keep track of the current receive capacity of the receiver (broker).

[0101] FIG. 6B expands on 606 of FIG. 6A and illustrates a method of transmitting one or more packets from a client to a broker via an HTTP tunnel connection according to one embodiment. As indicated at 620, the transmitted HTTP packets may optionally pass through a proxy server. As indicated at 622, the transmitted HTTP packets may optionally pass through one or more firewalls. A Web server may then receive the one or more HTTP packets as indicated at 624. For example, the client may transmit the packets to a proxy server, the proxy server may transmit the packets through a firewall onto the Internet, and the packets may be received through another firewall by the Web server.

[0102] The Web server may then forward the received HTTP packets to the broker as indicated at 626. In one embodiment, the packets may be transmitted to the broker via a TCP connection between the Web server and the broker that serves as one segment of the HTTP tunnel connection. In one embodiment, an HTTP tunnel servlet on the Web server node may serve as an interface between the Web server and the TCP connection to the broker. In one embodiment, one Web server and HTTP tunnel servlet may be used by two or more clients to communicate to a broker via HTTP tunnel connections. In this embodiment, the HTTP tunnel servlet may multiplex HTTP packets from the two or more clients onto the TCP connection. In one embodiment, the HTTP tunnel servlet may extract the messaging system message information from received HTTP packets and send only the message information to the broker over the TCP connection.

[0103] As indicated at 628, the broker may receive the HTTP packets from the Web server. In one embodiment, the broker may receive the packets on a TCP connection to the Web server. In one embodiment, a broker-side HTTP tunnel driver may receive the packets. In one embodiment, the broker may extract the messaging system messages from the HTTP packets and store the messages in a broker-side receive buffer. In another embodiment, the entire HTTP packet may be stored in a receive buffer. In one embodiment, there may be one receive buffer on the broker for each HTTP tunnel connection. In another embodiment, a single receive buffer may be used for two or more HTTP tunnel connections.

[0104] FIG. 6C expands on 608 of FIG. 6A and illustrates a method of a broker acknowledging to a client the receipt of HTTP packets from the client via the HTTP tunnel connection according to one embodiment. As indicated at 640, the broker may generate and send one or more acknowledgement (ACK) packets to the Web server. In one embodiment, the broker may store ACK packets in a broker-side transmit buffer. In one embodiment, the broker may send one ACK packet for each received HTTP packet. In another embodiment, the broker may send one ACK packet for each completely received messaging system message. In one embodiment, the ACK packets may be sent to the Web server over a TCP connection. In one embodiment, a broker-side HTTP tunnel driver may handle the transmission of the ACK packets on the TCP connection. In one embodiment, an HTTP tunnel servlet may receive the packets for the Web server.

[0105] As indicated at 642, the Web server may store the acknowledgement packets in an HTTP packet buffer. In one embodiment, there may be one HTTP packet buffer for each HTTP tunnel connection supported by the Web server. In another embodiment, there may be two HTTP packet buffers per tunnel connection, with one HTTP packet buffer for each message flow direction on each tunnel connection. In yet another embodiment, one or more HTTP packet buffers may be used for two or more tunnel connections.

[0106] As indicated at 644, at some point, the client may send an HTTP pull request packet to the Web server. In one embodiment, the client may periodically send HTTP pull requests to the Web server. In one embodiment, a separate thread on the client may handle periodically sending pull requests. The Web server may transmit to the client the one or more HTTP ACK packets stored in the HTTP packet buffer associated with the client in response to receiving the pull request packet as indicated at 646. As indicated at 648, the transmitted ACK packets may optionally pass through one or more firewalls. As indicated at 750, the transmitted ACK packets may optionally pass through a proxy server. As indicated at 652, the client may then receive the one or more ACK packets. In one embodiment, each ACK packet may include information on available space in the broker-side receive buffer to be used in flow control of HTTP packets from the client to the broker. The ACK packets may serve to acknowledge the receipt of the transmitted data so that the sender may free its transmit buffers.

[0107] FIGS. 7A-7C are flowcharts illustrating a method of sending messages from a messaging system client to a messaging system broker over an HTTP tunnel connection layer according to one embodiment. In FIG. 7A, a transport protocol tunnel connection between a client and a broker in a messaging system may be established as indicated at 700. As indicated at 702, one or more messaging system messages may be generated on the broker. In one embodiment, the generated messages may then be stored in a broker-side transmit buffer. One or more transport protocol packets encapsulating the one or more messages may be generated on the broker as indicated at 704. In one embodiment, a broker-side HTTP tunnel connection driver may generate the HTTP packets and include the messages as payloads of the packets. As indicated at 706, the one or more HTTP packets may then be transmitted to the client via the HTTP tunnel

connection. In one embodiment, the broker-side HTTP tunnel connection driver may handle the transmission of the packets.

[0108] As indicated at 708, the client may receive the HTTP packets, and then may acknowledge receipt of the one or more packets by sending one or more acknowledgement (ACK) packets to the broker via the HTTP tunnel connection. In one embodiment, the ACK packets may each include the currently available space in the client-side receive buffer. Thus, the sender (broker) can keep track of the current receive capacity of the receiver (client).

[0109] FIG. 7B expands on 706 of FIG. 7A and illustrates a method of transmitting one or more packets from a broker to a client via an HTTP tunnel connection according to one embodiment. As indicated at 720, the broker may generate and send one or more HTTP packets to the Web server. In one embodiment, the broker may store the HTTP packets in a broker-side transmit buffer. In one embodiment, the HTTP packets may be sent to the Web server over a TCP connection. In one embodiment, a broker-side HTTP tunnel driver may handle the transmission of the HTTP packets on the TCP connection.

[0110] As indicated at 722, the Web server may receive the HTTP packets and store the received HTTP packets in an HTTP packet buffer. In one embodiment, an HTTP tunnel servlet may receive the packets for the Web server. As indicated at 724, at some point, the client may send an HTTP pull request packet to the Web server. In one embodiment, the client may periodically send HTTP pull requests to the Web server. The Web server may transmit to the client the one or more HTTP packets stored in the HTTP packet buffer associated with the client in response to receiving the pull request packet as indicated at 726. As indicated at 728, the transmitted HTTP packets may optionally pass through one or more firewalls. As indicated at 730, the transmitted HTTP packets may optionally pass through a proxy server. As indicated at 732, the client may then receive the one or more HTTP packets. In one embodiment, the client may store the received HTTP packets in a client-side receive buffer. In one embodiment, each HTTP packet may include sequence information configured for use by the client in processing received messages in sequence.

[0111] FIG. 7C expands on 708 of FIG. 7A and illustrates a method of a client acknowledging to a broker the receipt of HTTP packets from the broker via the HTTP tunnel connection according to one embodiment. As indicated at 740, the client may generate and send one or more acknowledgement (ACK) packets to the broker. As indicated at 742, the transmitted ACK packets may optionally pass through a proxy server. As indicated at 744, the transmitted ACK packets may optionally pass through one or more firewalls. A Web server may then receive the one or more ACK packets as indicated at 746. For example, the client may transmit the ACK packets to a proxy server, the proxy server may transmit the ACK packets through a firewall onto the Internet, and the ACK packets may be received through another firewall by the Web server.

[0112] The Web server may then forward the received ACK packets to the broker as indicated at 748. In one embodiment, the ACK packets may be transmitted to the broker via a TCP connection between the Web server and the broker that serves as one segment of the HTTP tunnel

connection. In one embodiment, an HTTP tunnel servlet on the Web server node may serve as an interface between the Web server and the TCP connection to the broker.

[0113] As indicated at 750, the broker may receive the ACK packets from the Web server. In one embodiment, the broker may receive the packets on a TCP connection to the Web server. In one embodiment, a broker-side HTTP tunnel driver may receive the ACK packets. In one embodiment, the received ACK packets may be stored in a broker-side receive buffer. In one embodiment, each ACK packet may include information on available space in the client-side receive buffer to be used in flow control of HTTP packets from the broker to the client. The methods as described in FIGS. 6A-6C and FIGS. 7A-7C may be implemented in software, hardware, or a combination thereof. The order of method may be changed, and various steps may be added, reordered, combined, omitted, modified, etc.

[0114] FIG. 8 illustrates an exemplary HTTP tunneling protocol packet format according to one embodiment. The HTTP tunnel drivers at either end of a connection may encode all their data packets using this packet format. Each messaging system message may be carried separately as a HTTP request or response payload. A messaging system message may be sent as the payload of a single packet or, alternatively, the message may be broken into parts and sent as the payload of two or more packets.

[0115] The packet format may include several fields. Version 500 may indicate a version number of the packet that may be used to indicate different releases of the HTTP tunnel connection layer software. In one embodiment, the version 500 may be a 16-bit field. Packet type 502 may indicate the type of the packet. This field may be used to indicate to the HTTP tunnel driver at the other end what to do with the contents of this packet. In one embodiment, the packet type field may be 16 bits. Size 504 may indicate the size of the entire packet including the header. In one embodiment, this may be a 32-bit field. Connection ID 506 may be a unique integer that may be used as connection identifier. This value may be assigned at the time of connection establishment. This field may be used by the server to distinguish between connections to multiple clients. In one embodiment, this may be a 32-bit field. Packet sequence number 508 may be used to ensure sequential delivery of data bytes and flow control. In one embodiment, each packet may be assigned a unique incremental sequence number by the sender. In one embodiment, this may be a 32-bit field. One embodiment may include a receiver window size 510 that may be used for flow control and may indicate the capacity of the receive side buffer. In one embodiment, this may be a 16-bit field. One embodiment may guarantee that the packets are either delivered correctly, or in case of an error, are not delivered at all. Some embodiments may not use checksum 512, and thus this field may be reserved.

[0116] One embodiment of the HTTP tunnel connection layer may use a connection establishment protocol that may be used to initialize the following connection state components:

[0117] The HTTP tunnel servlet 214 allocates a unique connection ID for the new connection. It also may allocate one or more buffers 250 for packet streams in both directions.

[0118] The HTTP tunnel drivers on both server and client side allocate and initialize the transmit buffers and receive buffers for packet streams in both directions.

[0119] The following are examples of HTTP tunneling protocol packets that may be used in the connection establishment protocol and are not intended to be limiting. Note that one skilled in the art will recognize that other packet formats may be used within the scope of the invention.

[0120] A client 200 may initiate a connection to a server 202 by sending the following information in an HTTP packet to the HTTP tunnel servlet 214:

[0121] URL parameters="?ServerName=<ServerIdString>&Type=connect" Packet Type=connection initialization packet (e.g. CONN_INIT_PACKET (1)).

[0122] Connection ID =0

[0123] The servlet 214 may allocate a unique connection ID for this connection and send it to the client 200 and the server 202 in HTTP packets including the following information:

[0124] Packet Type=connection initialization packet (e.g. CONN_INIT_PACKET (1)).

[0125] ConnectionID=<connid>

[0126] This information may be sent to the client 200 as a response payload for the HTTP request that carried the client's connection initialization packet.

[0127] The server 202 may acknowledge the connection initialization packet to the client 200 in an ACK packet that may include the following information:

[0128] Pull URL parameters=

[0129] "?ServerName=<ServerIdString>&Type=pull&ConnId=<connid>"

[0130] Packet Type=

[0131] connection initialization acknowledgement (e.g. CONN_INIT_ACK (2)).

[0132] After completion, both client 200 and server 202 are aware of the newly established connection and normal data exchange can begin.

[0133] Once the connection is established, both sides may start sending data and connection management packets. The following are examples of HTTP tunneling protocol packets that may be used as data and connection management packets and are not intended to be limiting. Note that one skilled in the art will recognize that other packet formats may be used within the scope of the invention.

[0134] From the client 200 to the HTTP tunnel servlet 214 to the server 202:

[0135] URL parameters="?ServerName=<ServerIdString>&Type=push"

[0136] From the server 202 to the HTTP tunnel servlet 214 to the client 200:

[0137] Pull URL parameters=

[0138] "?ServerName=<ServerIdString>&Type=pull&ConnId=<connid>"

[0139] Each outgoing data packet (e.g. Packet Type=DATA_PACKET) may be assigned an incremental sequence number by the sender. The receiver may check the packet sequence number with its receive window. Any duplicate packets may be discarded.

[0140] After consuming a packet, the receiver may acknowledge the highest contiguous sequence number by sending the ACK packet as follows:

[0141] Packet Type=acknowledgement (e.g. ACK)

[0142] Connection ID=<connid>

[0143] Sequence=<sequence number of the data packet being acknowledged>

[0144] Receive Window Size=<remaining receive window capacity>

[0145] When an acknowledgement is received, the sender may update the round trip time for the connection. In one embodiment, a simple linear function of the computed round trip time may be used as the packet retransmission interval.

[0146] In one embodiment, when an acknowledgement packet reports a "Receive Window Size" of zero, the sender may stop sending further packets. The sender may send periodic repeat transmissions of the next packet to force the receiver to send a window update as soon as it is ready to receive more data. When the receiver indicates that it is ready to receive more data, the sender may resume sending packets.

[0147] In one embodiment, the HTTP Tunneling protocol may support the following runtime connection option. The client pull period is the duration (e.g. in seconds) of the idle period between consecutive pull requests sent by the client. If this value is positive, whenever the HTTP tunnel driver on the client side receives an empty response to its Pull request, the driver may sleep for the specified period before issuing another pull request. This may help conserve the servlet 214's resources and hence improve connection scalability.

[0148] Either client 200 or server 202 may set connection options. This may be used to control the runtime parameters associated with a connection. For example, this may be used to control how often packets are pulled from a Web server. As another example, this may be used to configure how long a client may remain inactive before the connection is dropped. The connection option values are part of the connection state information, and hence, in one embodiment, an HTTP packet including the following information may be used to update the connection options:

[0149] Packet Type=connection option packet (e.g. CONN_OPTION_PACKET)

Data = Option Type (integer)
 Option Value (integer)

[0150] In one embodiment, either end (client or server) may initiate connection shutdown by sending a connection close packet (e.g. Packet Type CONN_CLOSE_PACKET). Until both parties complete the connection shutdown, this packet may otherwise be treated like a normal data packet. This may help to ensure that all the data packets that are in

the pipeline ahead of the connection close packet are processed before the connection resources are destroyed. The remote end may consume all the data and acknowledge the connection close packet, at which point the connection is terminated and all the resources may be freed.

[0151] In one embodiment, a connection abort packet may be generated by the servlet when it realizes that either the server 202 or the client 200 has terminated ungracefully. The server termination may result in an IOException on the TCP connection between the servlet 214 and the server 202, and hence may be detected. In one embodiment, detecting client 200 termination may be handled as follows. If the servlet 214 does not receive a "pull" request from client 200 for a certain period of time, the servlet 214 may assume that the client 200 is not responding and thus may be down. The ungraceful connection shutdown may be achieved by sending a single packet with Packet Type connection abort packet (e.g. CONN_ABORT_PACKET) to the server 202 and possibly to the client 200 as well.

[0152] The following describes the initialization of the link between server 200 and HTTP tunneling servlet 214 according to one embodiment. The HTTP tunneling servlet 214 may listen on a fixed port number for TCP connections from servers. After the TCP connection is established to server 200, the server 200 may send the following information in an HTTP tunneling protocol packet to the servlet 214:

[0153] Packet Type: link initialization packet (e.g. LINK_INIT_PACKET)

ServerIdString (String)
ConnectionCount (Integer)
Data = Sequence of {ConnectionID (integer),
 ConnectionPullPeriod (integer)}

[0154] The ServerIdString may be used to establish the identity of the server 200. Clients may use this string to specify which server they want to talk to. This allows the use of a single servlet 214 as a gateway for multiple servers. The data portion of this packet may include information about any existing HTTP Tunnel connections. This allows HTTP tunnel connections to survive unexpected Web server/servlet engine failures or restarts.

[0155] Various embodiments may further include receiving, sending or storing instructions and/or data implemented in accordance with the foregoing description upon a carrier medium. Generally speaking, a carrier medium may include storage media or memory media such as magnetic or optical media, e.g., disk or CD-ROM, volatile or non-volatile media such as RAM (e.g. SDRAM, DDR SDRAM, RDRAM, SRAM, etc.), ROM, etc. as well as transmission media or signals such as electrical, electromagnetic, or digital signals, conveyed via a communication medium such as network and/or a wireless link.

[0156] In summary, a system and method for providing HTTP tunnel connections between entities such as clients and servers in a messaging system have been disclosed. It will be appreciated by those of ordinary skill having the benefit of this disclosure that the illustrative embodiments described above are capable of numerous variations without

departing from the scope and spirit of the invention. Various modifications and changes may be made as would be obvious to a person skilled in the art having the benefit of this disclosure. It is intended that the following claims be interpreted to embrace all such modifications and changes and, accordingly, the specifications and drawings are to be regarded in an illustrative rather than a restrictive sense.

What is claimed is:

1. A method comprising:

establishing a transport protocol tunnel connection from a first node in a messaging system to a second node in the messaging system;

generating a messaging system message on the first node;

generating one or more transport protocol packets, wherein the one or more transport protocol packets each includes at least a part of the messaging system message; and

transmitting the one or more transport protocol packets to the second node via the transport protocol tunnel connection;

wherein the transport protocol tunnel connection provides full-duplex transmission of messaging system messages between the first node and the second node, and wherein the transport protocol tunnel connection further provides delivery of the messaging system messages in the sequence in which the messaging system messages are generated.

2. The method as recited in claim 1, further comprising storing the messaging system message in a transmit buffer on the first node after said generating the messaging system message on the first node.

3. The method as recited in claim 1, wherein the transport protocol tunnel connection passes through a proxy server.

4. The method as recited in claim 3, wherein said transmitting the one or more transport protocol packets to the second node via the transport protocol tunnel connection comprises:

transmitting the one or more transport protocol packets from the first node to the proxy server; and

transmitting the one or more transport protocol packets from the proxy server to the second node.

5. The method as recited in claim 1, wherein the transport protocol tunnel connection passes through at least one firewall.

6. The method as recited in claim 1, wherein the transport protocol tunnel connection is established through a network.

7. The method as recited in claim 6, wherein the network is the Internet.

8. The method as recited in claim 1, wherein the first node is a client in the messaging system, and wherein the second node is a broker in the messaging system.

9. The method as recited in claim 1, wherein the transport protocol tunnel connection passes through a third node, and wherein, in said transmitting the one or more transport protocol packets to the second node, the method further comprises:

transmitting the one or more transport protocol packets to the third node; and

the third node forwarding the one or more transport protocol packets to the second node.

10. The method as recited in claim 9, wherein the one or more transport protocol packets are forwarded to the second node via a Transmission Control Protocol (TCP) connection portion of the transport protocol tunnel connection between the third node and the second node.

11. The method as recited in claim 9, wherein the third node is a Web server.

12. The method as recited in claim 1, wherein the transport protocol tunnel connection passes through a proxy server and a Web server, and wherein said transmitting the one or more transport protocol packets to the second node via the transport protocol tunnel connection comprises:

transmitting the one or more transport protocol packets from the first node to the proxy server;

transmitting the one or more transport protocol packets from the proxy server to the Web server; and

the Web server forwarding the one or more transport protocol packets to the second node.

13. The method as recited in claim 12, wherein the transport protocol tunnel connection passes through at least one firewall between the proxy server and the Web server.

14. The method as recited in claim 1, wherein the one or more transport protocol packets include messaging system message sequence information configured for use in processing two or more messaging system messages in sequence.

15. The method as recited in claim 1, further comprising:

receiving the transmitted one or more transport protocol packets on the second node; and

storing the messaging system message from the one or more transport protocol packets in a receive buffer on the second node.

16. The method as recited in claim 1, further comprising:

receiving the transmitted one or more transport protocol packets on the second node;

the second node generating an acknowledgement transport protocol packet to indicate successful receipt of the one or more transport protocol packets including the messaging system message; and

transmitting the acknowledgement transport protocol packet to the first node via the transport protocol tunnel connection.

17. The method as recited in claim 16, further comprising:

storing the messaging system message from the received one or more transport protocol packets in a receive buffer on the second node;

wherein the acknowledgement transport protocol packet includes information indicating available space in the receive buffer, and wherein the information indicating available space in the receive buffer is configured for use in flow control of messaging system messages transmitted from the first node to the second node.

18. The method as recited in claim 17, further comprising:

receiving the transmitted acknowledgement transport protocol packet on the first node;

generating one or more messaging system messages on the first node;

storing the one or more messaging system messages in a transmit buffer on the first node;

determining from the information indicating available space in the receive buffer included in the received acknowledgement transport protocol packet if there is space available to receive the one or more messaging system messages on the second node;

if said determining indicates there is space available to store the one or more messaging system messages in the receive buffer of the second node:

generating a second one or more transport protocol packets, wherein the second one or more transport protocol packets include the one or more messaging system messages; and

transmitting the second one or more transport protocol packets to the second node via the transport protocol tunnel connection; and

if said determining indicates there is not space available to store the second messaging system message in the receive buffer of the second node, inhibiting generating the second one or more transport protocol packets including the one or more messaging system messages.

19. The method as recited in claim 18, further comprising:

the first node receiving a transport protocol packet indicating available space in the receive buffer of the second node;

determining from the information indicating available space in the receive buffer included in the received transport protocol packet that there is space available to receive the one or more messaging system messages on the second node;

generating the second one or more transport protocol packets, wherein the second one or more transport protocol packets include the one or more messaging system messages; and

transmitting the second one or more transport protocol packets to the second node via the transport protocol tunnel connection.

20. The method as recited in claim 16, wherein the transport protocol tunnel connection passes through a third node, and wherein, in said transmitting the acknowledgement transport protocol packet to the first node, the method further comprises:

transmitting the acknowledgement transport protocol packet to the third node; and

storing the acknowledgement transport protocol packet in a transport protocol packet buffer on the third node.

21. The method as recited in claim 20, wherein, in said transmitting the acknowledgement transport protocol packet to the first node, the method further comprises:

the first node transmitting a transport protocol request packet to the third node; and

the third node transmitting the acknowledgement transport protocol packet stored in the transport protocol

packet buffer to the first node via the transport protocol tunnel connection in response to the transport protocol request packet.

22. The method as recited in claim 21, wherein the acknowledgement transport protocol packet is transmitted to the third node via a Transmission Control Protocol (TCP) connection portion of the transport protocol tunnel connection.

23. The method as recited in claim 16, wherein the transport protocol tunnel connection passes through a third node, and wherein, in said transmitting the acknowledgement transport protocol packet to the first node, the method further comprises:

transmitting the acknowledgement transport protocol packet to the third node; and

the third node forwarding the acknowledgement transport protocol packet to the first node.

24. The method as recited in claim 23, wherein the acknowledgement transport protocol packet are forwarded to the first node via a Transmission Control Protocol (TCP) connection portion of the transport protocol tunnel connection.

25. The method as recited in claim 1, wherein the first node is a server in the messaging system, and wherein the second node is a client in the messaging system.

26. The method as recited in claim 1, wherein the transport protocol tunnel connection passes through a third node, and wherein, in said transmitting the one or more transport protocol packets to the second node via the transport protocol tunnel connection, the method further comprises:

transmitting the one or more transport protocol packets to the third node; and

storing the one or more transport protocol packets in a transport protocol packet buffer on the third node.

27. The method as recited in claim 26, wherein, in said transmitting the one or more transport protocol packets to the second node via the transport protocol tunnel connection, the method further comprises:

the second node sending one or more transport protocol request packets to the third node; and

the third node transmitting the one or more transport protocol packets stored in the transport protocol packet buffer to the second node via the transport protocol tunnel connection in response to the one or more transport protocol request packets.

28. The method as recited in claim 26, wherein the third node is a Web server.

29. The method as recited in claim 1, wherein the transport protocol is Hypertext Transport Protocol (HTTP).

30. The method as recited in claim 1, wherein the transport protocol is one of UDP (User Datagram Protocol), IrDA (Infrared Data Association), SNA (Systems Network Architecture), IPX (Internetwork Packet eXchange), and Bluetooth.

31. A method comprising:

establishing a Hypertext Transport Protocol (HTTP) tunnel connection from a first node in a messaging system to a second node in the messaging system;

generating a messaging system message on the first node;

generating one or more HTTP packets, wherein the one or more HTTP packets each includes at least a part of the messaging system message; and

transmitting the one or more HTTP packets to the second node via the HTTP tunnel connection;

wherein the HTTP tunnel connection provides full-duplex transmission of messaging system messages between the first node and the second node, and wherein the HTTP tunnel connection further provides delivery of the messaging system messages in the sequence in which the messaging system messages are generated.

32. The method as recited in claim 31, wherein the first node is a client in the messaging system, wherein the HTTP tunnel connection passes through a proxy server, and wherein said transmitting the one or more HTTP packets to the second node via the HTTP tunnel connection comprises:

transmitting the one or more HTTP packets from the client to the proxy server; and

transmitting the one or more HTTP packets from the proxy server to the second node.

33. The method as recited in claim 31, wherein the HTTP tunnel connection is established through the Internet, and wherein the HTTP tunnel connection passes through at least one firewall.

34. The method as recited in claim 31, wherein the first node is a client in the messaging system, and wherein the second node is a broker in the messaging system.

35. The method as recited in claim 31, wherein the HTTP tunnel connection passes to through a Web server, wherein the second node is a broker in the messaging system, and wherein, in said transmitting the one or more HTTP packets to the second node, the method further comprises:

transmitting the one or more HTTP packets to the Web server; and

the Web server forwarding the one or more HTTP packets to the broker via a Transmission Control Protocol (TCP) connection portion of the HTTP tunnel connection between the Web server and the broker.

36. The method as recited in claim 31, wherein the first node is a client in the messaging system, wherein the second node is a broker in the messaging system, wherein the HTTP tunnel connection passes through a proxy server and a Web server, and wherein said transmitting the one or more HTTP packets to the broker via the HTTP tunnel connection comprises:

transmitting the one or more HTTP packets from the client to the proxy server;

transmitting the one or more HTTP packets from the proxy server to the Web server; and

the Web server forwarding the one or more HTTP packets to the broker;

wherein the HTTP tunnel connection passes through at least one firewall between the proxy server and the Web server.

37. The method as recited in claim 31, wherein the one or more HTTP packets include messaging system message sequence information configured for use in processing two or more messaging system messages in sequence.

38. The method as recited in claim 31, further comprising: receiving the transmitted one or more HTTP packets on the second node;

storing the messaging system message from the one or more HTTP packets in a receive buffer on the second node;

the second node generating an acknowledgement HTTP packet to indicate successful receipt of the one or more HTTP packets including the messaging system message; and

transmitting the acknowledgement HTTP packet to the first node via the HTTP tunnel connection.

39. The method as recited in claim 38, wherein the acknowledgement HTTP packet includes information indicating available space in the receive buffer, the method further comprising:

receiving the transmitted acknowledgement HTTP packet on the first node;

generating one or more messaging system messages on the first node;

storing the one or more messaging system messages in a transmit buffer on the first node;

determining from the information indicating available space in the receive buffer included in the received acknowledgement HTTP packet that there is not space available to receive the one or more messaging system messages on the second node;

the first node receiving an HTTP packet from the second node indicating available space in the receive buffer of the second node;

determining from the information indicating available space in the receive buffer included in the received HTTP packet that there is space available to receive the one or more messaging system messages on the second node;

generating a second one or more HTTP packets, wherein the second one or more HTTP packets include the one or more messaging system messages; and

transmitting the second one or more HTTP packets to the second node via the HTTP tunnel connection.

40. The method as recited in claim 38, wherein the first node is a client in the messaging system, wherein the HTTP tunnel connection passes through a Web server, and wherein, in said transmitting the acknowledgement HTTP packet to the first node, the method further comprises:

transmitting the acknowledgement HTTP packet to the Web server;

storing the acknowledgement HTTP packet in an HTTP packet buffer on the Web server;

the client sending an HTTP request packet to the Web server; and

the Web server transmitting the acknowledgement HTTP packet stored in the HTTP packet buffer to the client via the HTTP tunnel connection in response to the HTTP request packet.

41. The method as recited in claim 38, wherein the first node is a broker in the messaging system, wherein the HTTP tunnel connection passes through a Web server, and wherein, in said transmitting the acknowledgement HTTP packet to the first node, the method further comprises:

transmitting the acknowledgement HTTP packet to the Web server; and

the Web server forwarding the acknowledgement HTTP packet to the first node via a Transmission Control Protocol (TCP) connection portion of the HTTP tunnel connection.

42. The method as recited in claim 31, wherein the first node is a server in the messaging system, and wherein the second node is a client in the messaging system.

43. The method as recited in claim 31, wherein the second node is a client in the messaging system, wherein the HTTP tunnel connection passes through a Web server, and wherein, in said transmitting the one or more HTTP packets to the second node via the HTTP tunnel connection, the method further comprises:

transmitting the one or more HTTP packets to the Web server;

storing the one or more HTTP packets in an HTTP packet buffer on the Web server;

the client sending one or more HTTP request packets to the Web server; and

the Web server transmitting the one or more HTTP packets stored in the HTTP packet buffer to the client via the HTTP tunnel connection in response to the one or more HTTP request packets.

44. A method comprising:

establishing a transport protocol tunnel connection from a first node in a messaging system to a second node in the messaging system;

generating a sequence of messaging system messages on the first node;

generating a plurality of transport protocol packets on the first node, wherein each of the transport protocol packets includes at least a part of one of the sequence of messaging system messages, and wherein each of the transport protocol packets includes sequence information for the particular messaging system message;

transmitting the plurality of transport protocol packets to the second node in the messaging system via the transport protocol tunnel connection;

receiving the plurality of transport protocol packets on the second node; and

processing the sequence of messaging system messages on the second node, wherein said processing uses the sequence information for the plurality of messaging system messages in the plurality of transport protocol packets.

45. The method as recited in claim 44, wherein the transport protocol tunnel connection is established through the Internet, and wherein the transport protocol tunnel connection passes through at least one firewall.

46. The method as recited in claim 44, wherein the first node is a client in the messaging system, wherein the second node is a broker in the messaging system.

47. The method as recited in claim 44, wherein the second node is a broker in the messaging system, wherein the transport protocol tunnel connection passes through a Web server, and wherein said transmitting the plurality of transport protocol packets to the second node in the messaging system via the transport protocol tunnel connection comprises:

transmitting the plurality of transport protocol packets from the first node to the Web server; and

the Web server forwarding the plurality of transport protocol packets to the broker;

wherein the transport protocol tunnel connection passes through at least one firewall between the proxy server and the Web server.

48. The method as recited in claim 44, wherein the first node is a broker in the messaging system, wherein the second node is a client in the messaging system.

49. The method as recited in claim 44, wherein the second node is a client in the messaging system, wherein the transport protocol tunnel connection passes through a Web server, and wherein, in said transmitting the plurality of transport protocol packets to the second node in the messaging system via the transport protocol tunnel connection comprises:

transmitting the plurality of transport protocol packets to the Web server;

storing the plurality of transport protocol packets in a transport protocol packet buffer on the Web server;

the client sending one or more transport protocol request packets to the Web server; and

the Web server transmitting the plurality of transport protocol packets stored in the transport protocol packet buffer to the client via the transport protocol tunnel connection in response to the one or more transport protocol request packets.

50. The method as recited in claim 44, further comprising:

storing the sequence of messaging system messages from the received transport protocol packets in a receive buffer on the second node;

the second node generating an acknowledgement transport protocol packet for each of the received transport protocol packets to indicate successful receipt of the transport protocol packets including the sequence of messaging system messages; and

transmitting the acknowledgement transport protocol packets to the first node via the transport protocol tunnel connection;

wherein each of the acknowledgement transport protocol packets includes information indicating available space in the receive buffer, wherein the information indicating available space in the receive buffer is configured for use in flow control of messaging system messages transmitted from the first node to the second node.

51. The method as recited in claim 44, wherein the transport protocol is Hypertext Transport Protocol (HTTP).

52. A method comprising:

establishing a transport protocol tunnel connection from a first node in a messaging system to a second node in the messaging system;

the first node receiving a first transport protocol packet from the second node indicating available space in a receive buffer of the second node;

generating one or more messaging system messages on the first node;

storing the one or more messaging system messages in a transmit buffer on the first node;

determining from the information indicating available space in the receive buffer included in the received acknowledgement HTTP packet that there is not space available to receive the one or more messaging system messages on the second node;

the first node receiving a second transport protocol packet from the second node indicating available space in the receive buffer of the second node;

determining from the information indicating available space in the receive buffer included in the received second transport protocol packet that there is space available to receive the one or more messaging system messages on the second node;

generating one or more transport protocol packets, wherein the second one or more transport protocol packets include the generated one or more messaging system messages; and

transmitting the one or more transport protocol packets to the second node via the transport protocol tunnel connection.

53. The method as recited in claim 52, wherein the transport protocol tunnel connection is established through the Internet, and wherein the transport protocol tunnel connection passes through at least one firewall.

54. The method as recited in claim 52, wherein the first node is a client in the messaging system, wherein the second node is a broker in the messaging system.

55. The method as recited in claim 52, wherein the first node is a broker in the messaging system, wherein the second node is a client in the messaging system.

56. The method as recited in claim 52, further comprising:

receiving the one or more transport protocol packets on the second node;

storing the one or more messaging system messages from the received one or more transport protocol packets in the receive buffer on the second node;

the second node generating one or more acknowledgement transport protocol packets to indicate successful receipt of the one or more transport protocol packets including the one or more of messaging system messages; and

transmitting the one or more acknowledgement transport protocol packets to the first node via the transport protocol tunnel connection;

wherein each of the acknowledgement transport protocol packets includes information indicating available space in the receive buffer.

57. The method as recited in claim 52, wherein the one or more transport protocol packets include messaging system message sequence information configured for use in processing two or more messaging system messages in sequence.

58. The method as recited in claim 52, wherein the transport protocol is Hypertext Transport Protocol (HTTP).

59. A messaging system comprising:

a first node comprising a first memory;

a second node comprising a second memory;

wherein the first memory comprises first program instructions executable within the first node to:

establish a transport protocol tunnel connection from the first node to the second node through a network;

generate a messaging system message;

generate one or more transport protocol packets, wherein the one or more transport protocol packets each includes at least a part of the messaging system message; and

transmit the one or more transport protocol packets to the second node via the transport protocol tunnel connection;

wherein the transport protocol tunnel connection provides full-duplex transmission of messaging system messages between the first node and the second node, and wherein the transport protocol tunnel connection further provides delivery of the messaging system messages in the sequence in which the messaging system messages are generated.

60. The messaging system as recited in claim 59, wherein the first node further comprises a transmit buffer, wherein the first program instructions are further executable within the first node to store the messaging system message in the transmit buffer on the first node after said generating the messaging system message.

61. The messaging system as recited in claim 59, wherein the transport protocol tunnel connection passes through a proxy server, and wherein, in said transmitting the one or more transport protocol packets to the second node via the transport protocol tunnel connection, the first program instructions are further executable within the first node to transmit the one or more transport protocol packets from the first node to the proxy server, wherein the proxy server is configured to transmit the one or more transport protocol packets to the second node.

62. The messaging system as recited in claim 59, wherein the transport protocol tunnel connection passes through at least one firewall.

63. The messaging system as recited in claim 59, wherein the transport protocol tunnel connection is established through the Internet.

64. The messaging system as recited in claim 59, wherein the first node is a client in the messaging system, and wherein the second node is a broker in the messaging system.

65. The messaging system as recited in claim 59, wherein the messaging system further comprises:

a third node comprising a third memory;

wherein the transport protocol tunnel connection passes through the third node, and wherein, in said transmitting the one or more transport protocol packets to the second node, the first program instructions are further executable within the first node to:

transmit the one or more transport protocol packets to the third node;

wherein the third memory comprises third program instructions executable within the third node to:

receive the one or more transport protocol packets from the first node; and

forward the one or more received transport protocol packets to the second node.

66. The messaging system as recited in claim 65, wherein the one or more transport protocol packets are forwarded from the third node to the second node via a Transmission Control Protocol (TCP) connection portion of the transport protocol tunnel connection between the third node and the second node.

67. The messaging system as recited in claim 66, wherein the transport protocol tunnel connection passes through at least one firewall between the first node and the third node.

68. The messaging system as recited in claim 59, wherein the one or more transport protocol packets include messaging system message sequence information configured for use in processing two or more messaging system messages in sequence.

69. The messaging system as recited in claim 59, wherein the second memory comprises second program instructions executable within the second node to:

receive the transmitted one or more transport protocol packets;

generate an acknowledgement transport protocol packet to indicate successful receipt of the one or more transport protocol packets including the messaging system message; and

transmit the acknowledgement transport protocol packet to the first node via the transport protocol tunnel connection.

70. The messaging system as recited in claim 69, wherein the second node further comprises a receive buffer, wherein the second program instructions are further executable within the second node to:

store the messaging system message from the received one or more transport protocol packets in the receive buffer of the second node;

wherein the acknowledgement transport protocol packet includes information indicating available space in the receive buffer of the second node.

71. The messaging system as recited in claim 70, wherein the first node further comprises a transmit buffer, wherein the first program instructions are further executable within the first node to:

receive the transmitted acknowledgement transport protocol packet;

generate one or more messaging system messages;

store the one or more messaging system messages in the transmit buffer on the first node;

from the information indicating available space in the receive buffer included in the received acknowledgement transport protocol packet, determine if there is space available to receive the one or more messaging system messages on the second node;

if said determining indicates there is space available to store the one or more messaging system messages in the receive buffer of the second node:

generate a second one or more transport protocol packets, wherein the second one or more transport protocol packets include the one or more messaging system messages; and

transmit the second one or more transport protocol packets to the second node via the transport protocol tunnel connection; and

if said determining indicates there is not space available to store the second messaging system message in the receive buffer of the second node, inhibit generating the second one or more transport protocol packets including the one or more messaging system messages.

72. The messaging system as recited in claim 71, wherein the first program instructions are further executable within the first node to:

receive a transport protocol packet indicating available space in the receive buffer of the second node;

from the information indicating available space in the receive buffer included in the received transport protocol packet, determine that there is space available to receive the one or more messaging system messages on the second node;

generate the second one or more transport protocol packets, wherein the second one or more transport protocol packets include the one or more messaging system messages; and

transmit the second one or more transport protocol packets to the second node via the transport protocol tunnel connection.

73. The messaging system as recited in claim 69, further comprising:

a third node comprising:

a third memory; and

a transport protocol packet buffer;

wherein the transport protocol tunnel connection passes through the third node, wherein the third memory comprises third program instructions executable within the third node to:

receive the acknowledgement transport protocol packet transmitted to the first node via the transport protocol tunnel connection from the second node; and

store the received acknowledgement transport protocol packet in the transport protocol packet buffer.

74. The messaging system as recited in claim 73, wherein the first program instructions are further executable within the first node to:

transmit a transport protocol request packet to the third node; and

wherein the third program instructions are further executable within the third node to:

receive the transport protocol request packet from the first node; and

transmit the acknowledgement transport protocol packet stored in the transport protocol packet buffer to the first node via the transport protocol tunnel connection in response to the received transport protocol request packet.

75. The messaging system as recited in claim 69, further comprising:

a third node comprising a third memory, wherein the transport protocol tunnel connection passes through the third node, wherein the third memory comprises third program instructions executable within the third node to:

receive the acknowledgement transport protocol packet transmitted to the first node via the transport protocol tunnel connection from the second node; and

forward the acknowledgement transport protocol packet to the first node.

76. The messaging system as recited in claim 59, wherein the first node is a server in the messaging system, and wherein the second node is a client in the messaging system.

77. The messaging system as recited in claim 59, further comprising:

a third node comprising:

a third memory; and

a transport protocol packet buffer;

wherein the transport protocol tunnel connection passes through the third node, wherein the third memory comprises third program instructions executable within the third node to:

receive the one or more transport protocol packets transmitted to the second node via the transport protocol tunnel connection from the first node; and

store the one or more transport protocol packets in the transport protocol packet buffer on the third node.

wherein the second program instructions are further executable within the second node to transmit one or more transport protocol request packets to the third node; and

wherein the third program instructions are further executable within the third node to:

receive the one or more transmitted transport protocol request packets; and

transmit the one or more transport protocol packets stored in the transport protocol packet buffer to the second node via the transport protocol tunnel connection in response to the received one or more transport protocol request packets.

78. The messaging system as recited in claim 59, wherein the transport protocol is Hypertext Transport Protocol (HTTP).

79. The messaging system as recited in claim 59, wherein the transport protocol is one of UDP (User Datagram Protocol), IrDA (Infrared Data Association), SNA (Systems Network Architecture), IPX (Internetwork Packet eXchange), and Bluetooth.

80. A carrier medium comprising program instructions, wherein the program instructions are computer-executable to implement:

establishing a transport protocol tunnel connection from a first node in a messaging system to a second node in the messaging system;

generating a messaging system message on the first node;

generating one or more transport protocol packets, wherein the one or more transport protocol packets each includes at least a part of the messaging system message; and

transmitting the one or more transport protocol packets to the second node via the transport protocol tunnel connection;

wherein the transport protocol tunnel connection provides full-duplex transmission of messaging system messages between the first node and the second node, and wherein the transport protocol tunnel connection further provides delivery of the messaging system messages in the sequence in which the messaging system messages are generated.

81. The carrier medium as recited in claim 80, wherein the transport protocol tunnel connection passes through a Web server, wherein the second node is a broker in the messaging system, and wherein, in said transmitting the one or more transport protocol packets to the second node, the program instructions are further computer-executable to implement:

transmitting the one or more transport protocol packets to the Web server; and

the Web server forwarding the one or more transport protocol packets to the broker via a Transmission Control Protocol (TCP) connection portion of the transport protocol tunnel connection between the Web server and the broker.

82. The carrier medium as recited in claim 80, wherein the first node is a client in the messaging system, wherein the second node is a broker in the messaging system, wherein the transport protocol tunnel connection passes through a proxy server and a Web server, and wherein, in said transmitting the one or more transport protocol packets to the broker via the transport protocol tunnel connection, the program instructions are further computer-executable to implement:

transmitting the one or more transport protocol packets from the client to the proxy server;

transmitting the one or more transport protocol packets from the proxy server to the Web server; and

the Web server forwarding the one or more transport protocol packets to the broker;

wherein the transport protocol tunnel connection passes through at least one firewall between the proxy server and the Web server.

83. The carrier medium as recited in claim 80, wherein the acknowledgement transport protocol packet includes information indicating available space in the receive buffer, and wherein the program instructions are further computer-executable to implement:

receiving the transmitted one or more transport protocol packets on the second node;

storing the messaging system message from the one or more transport protocol packets in a receive buffer on the second node;

generating on the second node an acknowledgement transport protocol packet to indicate successful receipt of the one or more transport protocol packets including the messaging system message;

transmitting the acknowledgement transport protocol packet to the first node via the transport protocol tunnel connection;

receiving the transmitted acknowledgement transport protocol packet on the first node;

generating one or more messaging system messages on the first node;

storing the one or more messaging system messages in a transmit buffer on the first node;

determining from the information indicating available space in the receive buffer included in the received acknowledgement transport protocol packet that there is space available to receive the one or more messaging system messages on the second node;

generating a second one or more transport protocol packets, wherein the second one or more transport protocol packets include the one or more messaging system messages; and

transmitting the second one or more transport protocol packets to the second node via the transport protocol tunnel connection.

84. The carrier medium as recited in claim 80, wherein the first node is a client in the messaging system, wherein the transport protocol tunnel connection passes through a Web server, and wherein the program instructions are further computer-executable to implement:

receiving the transmitted one or more transport protocol packets on the second node;

storing the messaging system message from the one or more transport protocol packets in a receive buffer on the second node;

the second node generating an acknowledgement transport protocol packet to indicate successful receipt of the one or more transport protocol packets including the messaging system message;

transmitting the acknowledgement transport protocol packet to the Web server;

storing the acknowledgement transport protocol packet in a transport protocol packet buffer on the Web server;

the client sending a transport protocol request packet to the Web server; and

the Web server transmitting the acknowledgement transport protocol packet stored in the transport protocol packet buffer to the client via the transport protocol tunnel connection in response to the transport protocol request packet.

85. The carrier medium as recited in claim 80, wherein the first node is a broker in the messaging system, wherein the transport protocol tunnel connection passes through a Web server, and wherein the program instructions are further computer-executable to implement:

receiving the transmitted one or more transport protocol packets on the second node;

storing the messaging system message from the one or more transport protocol packets in a receive buffer on the second node;

the second node generating an acknowledgement transport protocol packet to indicate successful receipt of the one or more transport protocol packets including the messaging system message;

transmitting the acknowledgement transport protocol packet to the Web server; and

the Web server forwarding the acknowledgement transport protocol packet to the first node via a Transmission Control Protocol (TCP) connection portion of the transport protocol tunnel connection.

86. The carrier medium as recited in claim 80, wherein the second node is a client in the messaging system, wherein the transport protocol tunnel connection passes through a Web server, and wherein, in said transmitting the one or more transport protocol packets to the second node via the transport protocol tunnel connection, the program instructions are further computer-executable to implement:

transmitting the one or more transport protocol packets to the Web server;

storing the one or more transport protocol packets in a transport protocol packet buffer on the Web server;

the client sending one or more transport protocol request packets to the Web server; and

the Web server transmitting the one or more transport protocol packets stored in the transport protocol packet buffer to the client via the transport protocol tunnel connection in response to the one or more transport protocol request packets.

87. The carrier medium as recited in claim 80, wherein the transport protocol is Hypertext Transport Protocol (HTTP).

* * * * *

(B) Evidence for Claims 4-9 – Relied Upon

The following item (1) listed below is hereby entered as evidence relied upon by the Examiner as to grounds of rejection for claims 4-9, to be reviewed on appeal. Also listed for each item is where said evidence was entered into the record by the Examiner.

(1) Copy of US Patent Application Number 20020083183 ("Pujare"). This evidence was entered into the record by the Examiner on page 4 paragraph 8. 9. 10. 11. 12. and page 5 at paragraph 13. of the Office Action mailed 08/12/2005.

Copies of all References follows.

//

1



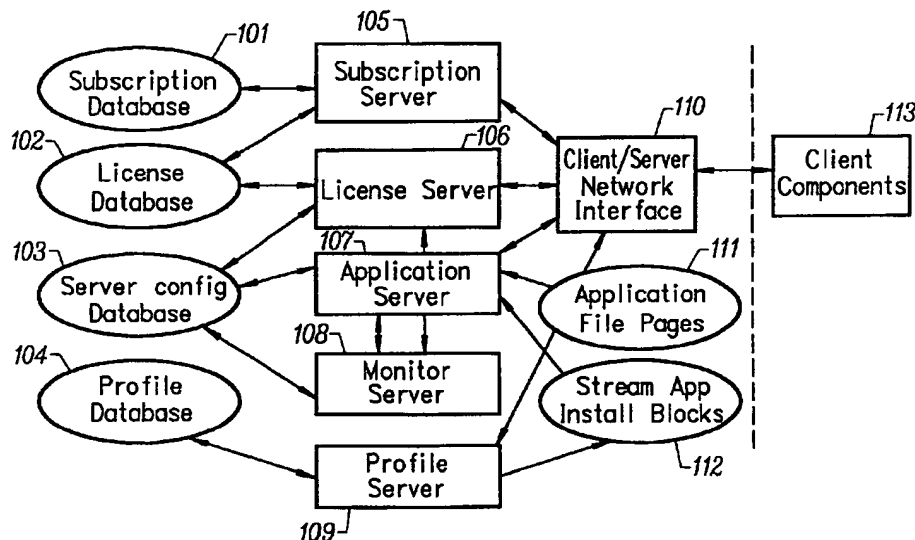
US 20020083183A1

(19) **United States**(12) **Patent Application Publication** (10) **Pub. No.: US 2002/0083183 A1**
(43) **Pub. Date: Jun. 27, 2002**(54) **CONVENTIONALLY CODED APPLICATION
CONVERSION SYSTEM FOR STREAMED
DELIVERY AND EXECUTION**(52) **U.S. Cl. 709/231; 709/224**(76) **Inventors: Sanjay Pujare, San Jose, CA (US);
Robert Deuel, Mountain View, CA
(US); Nicholas Ryan, Santa Clara, CA
(US); Manuel Benitez, Cupertino, CA
(US); David Lin, Mountain View, CA
(US)**

Correspondence Address:
**GLENN PATENT GROUP
3475 EDISON WAY
SUITE L
MENLO PARK, CA 94025 (US)**

(21) **Appl. No.: 09/826,607**(22) **Filed: Apr. 5, 2001****Related U.S. Application Data**(63) **Non-provisional of provisional application No.
60/246,384, filed on Nov. 6, 2000.****Publication Classification**(51) **Int. Cl.⁷ G06F 15/173; G06F 15/16**(57) **ABSTRACT**

A conventionally coded application conversion system for streamed delivery and execution converts locally installable applications into a data set suitable for streaming over a network. The invention monitors two classes of information during an application installation on a local computer system. System registry modifications are monitored and the modification data are recorded when the installation program writes to the registry of the local computer system. File modification data are logged each time an installation program modifies a file on the system. This data is used to create an initialization data set which is the first set of data to be streamed from the server to the client and contains the information captured needed by the client to prepare the client machine for streaming a particular application. A runtime data set is also created that contains the rest of the data that is streamed to the client once the client machine is initialized for a particular application. A versioning table contains a list of root file numbers and version numbers which are used to track application patches and upgrades. The invention monitors a running application that is being configured for a particular working environment on the local computer system. The data acquired are used to duplicate the same configuration on multiple client machines.

Server Components Supporting Application Delivery & Execution License

Server Components Supporting Application Delivery & Execution License

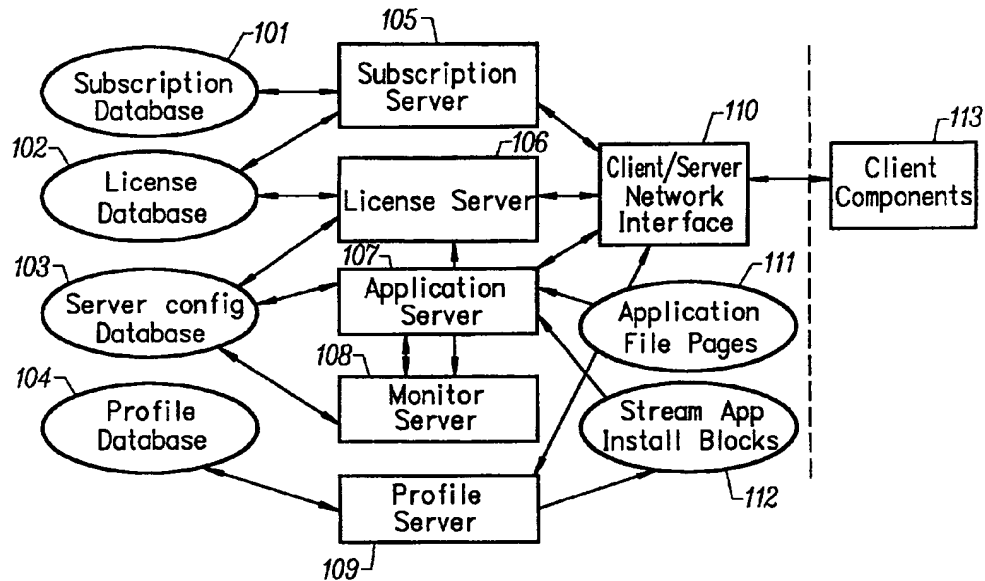


FIG. 1

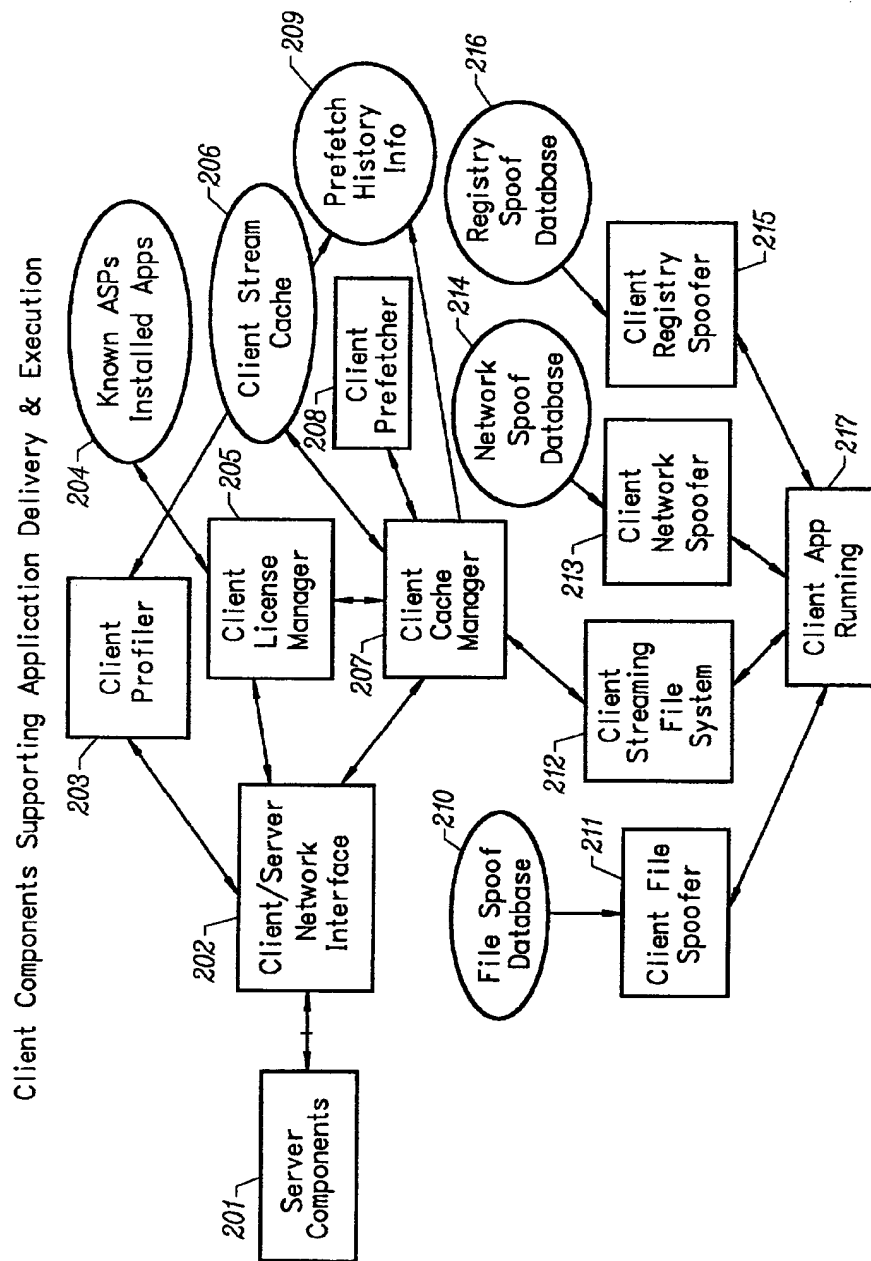


FIG. 2

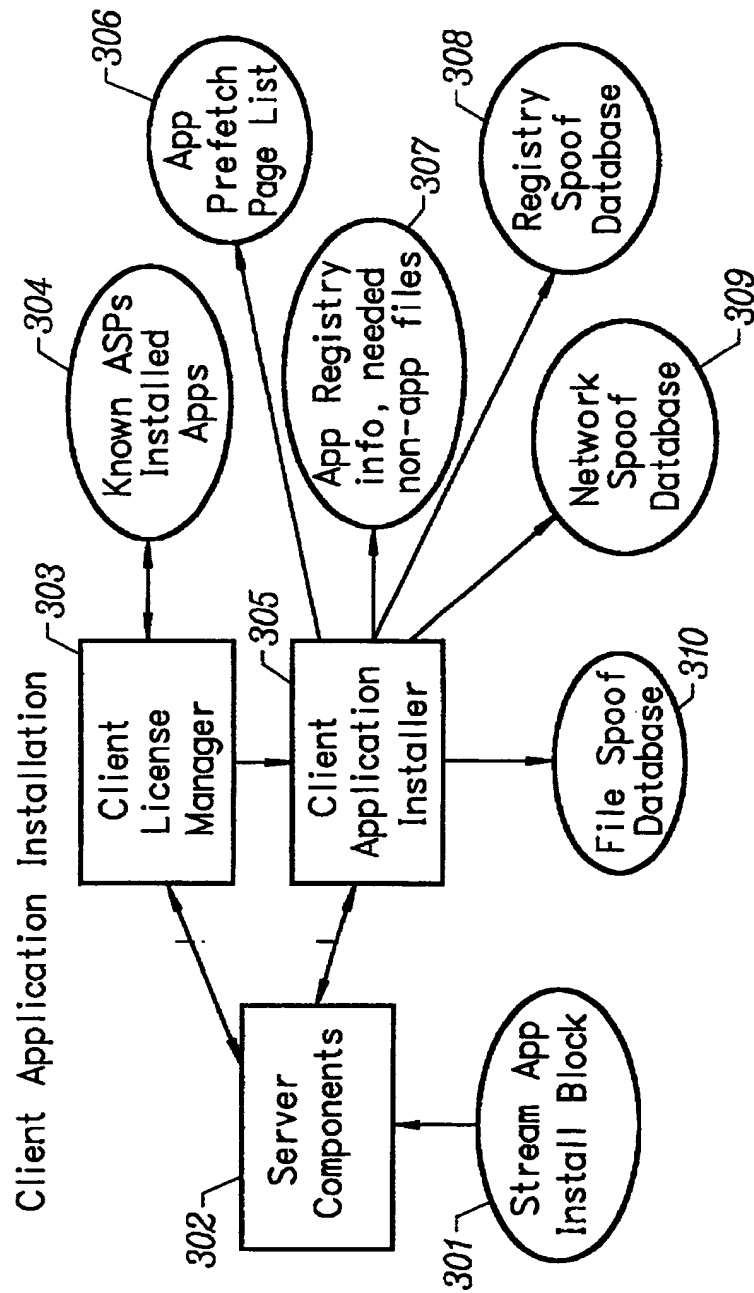


FIG. 3

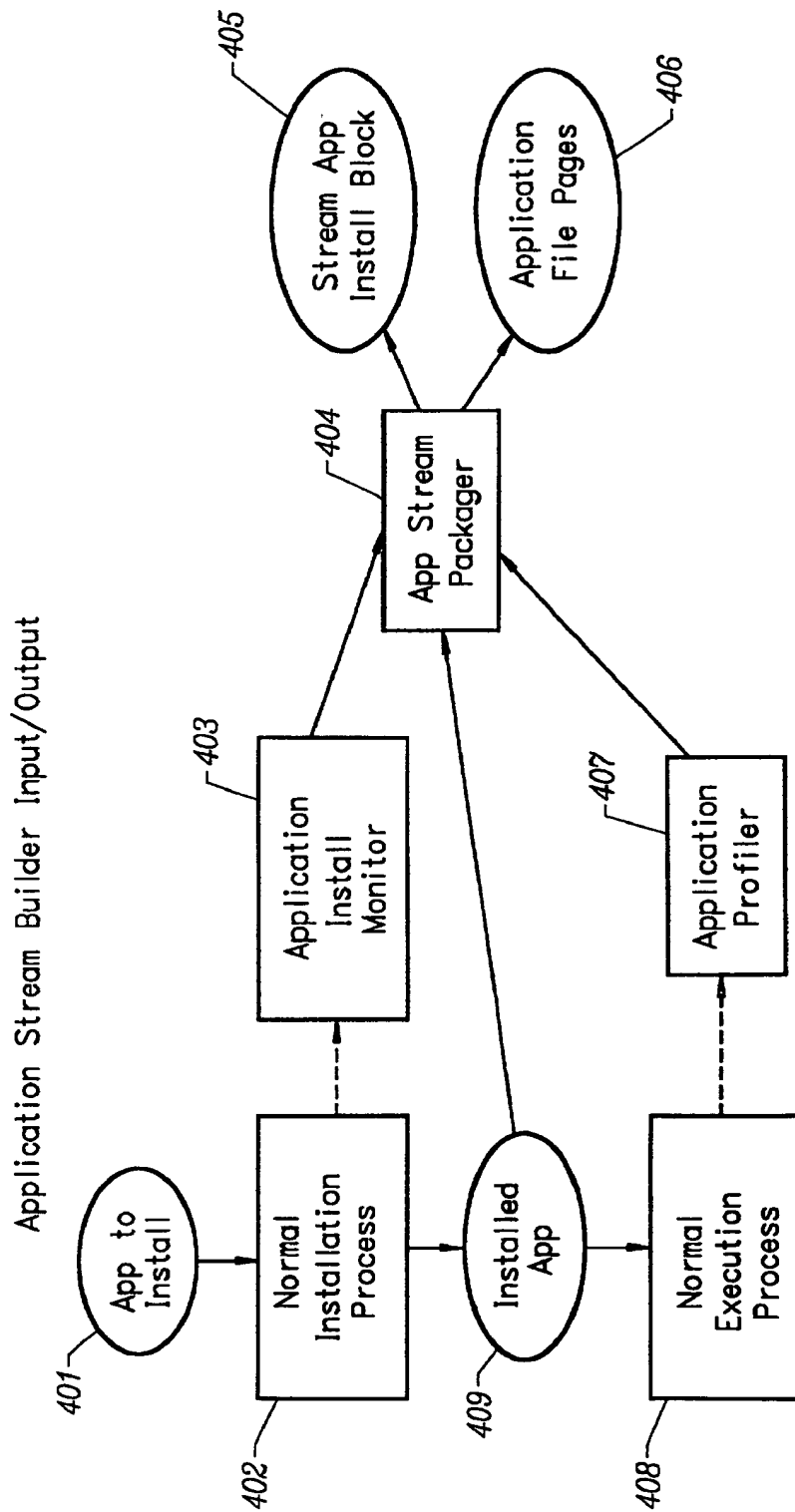


FIG. 4

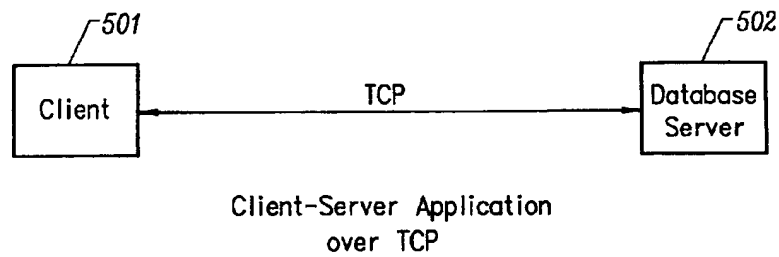


FIG. 5A

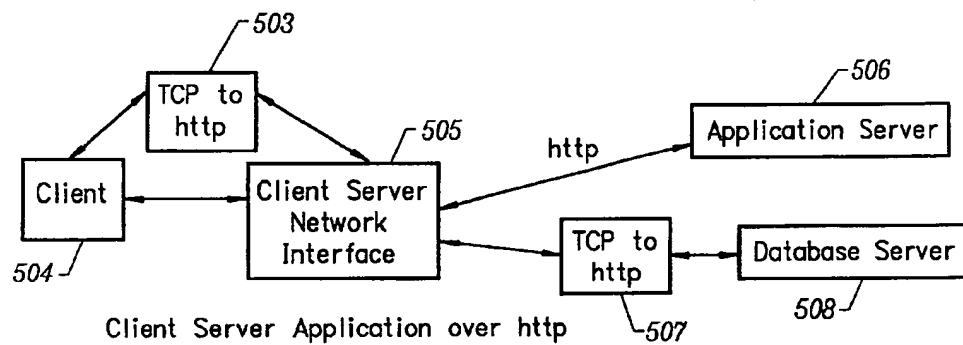


FIG. 5B

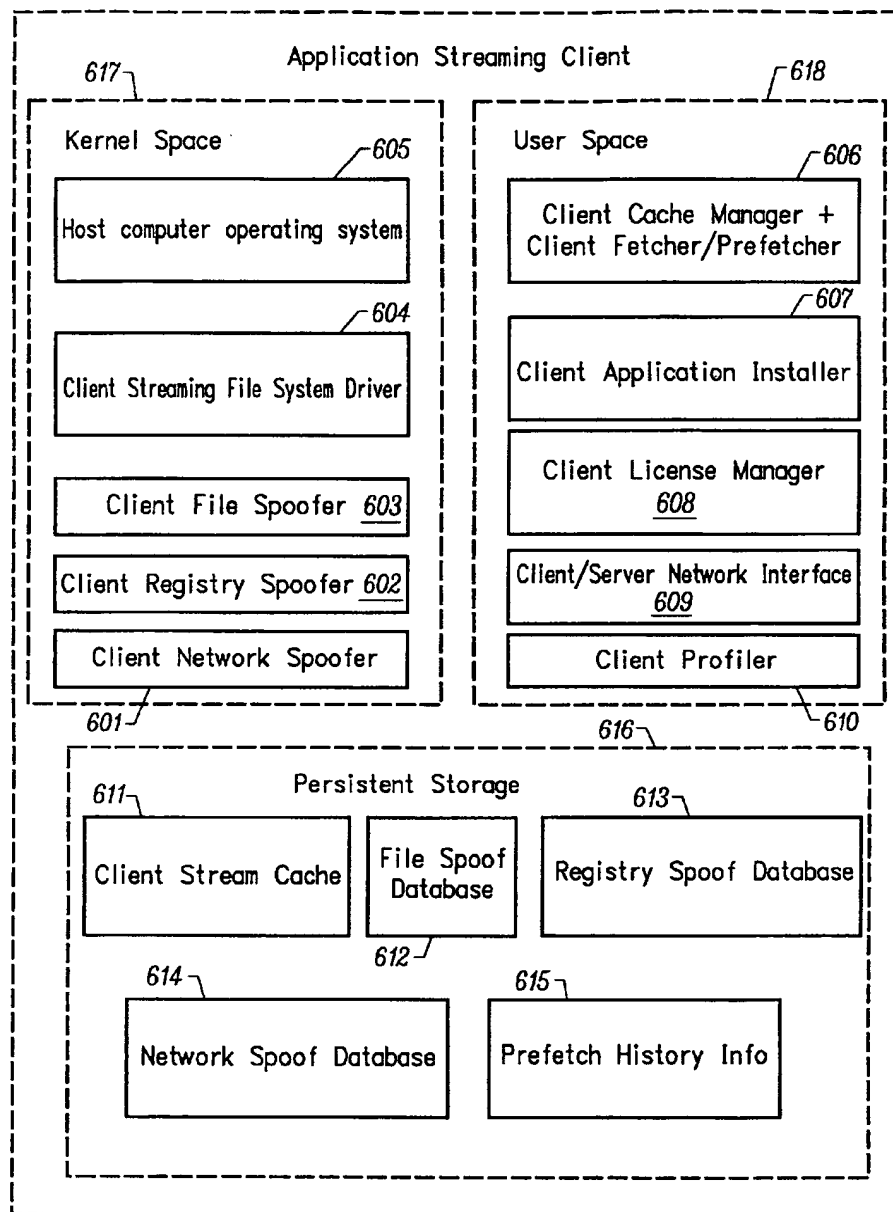


FIG. 6A

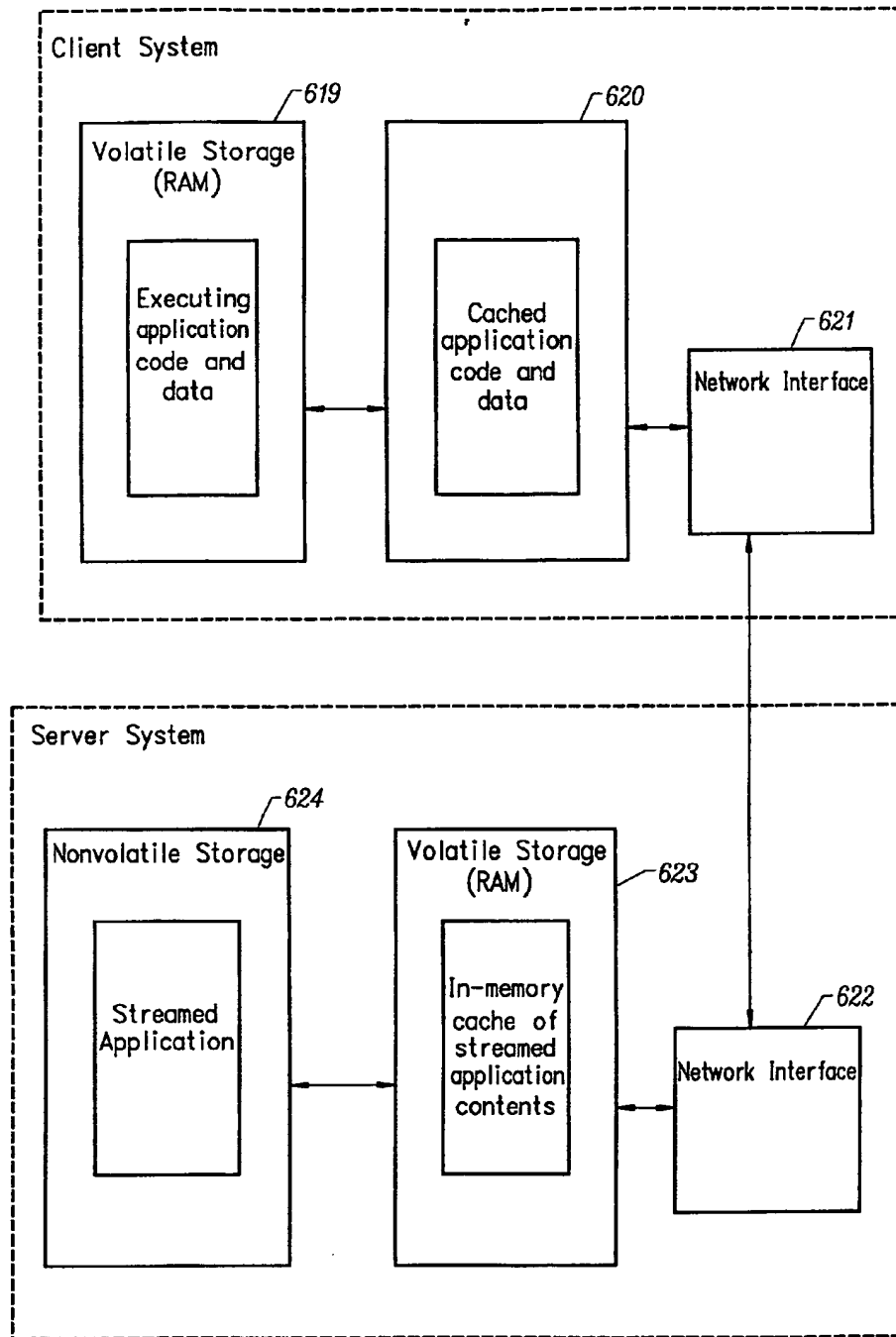


FIG. 6B

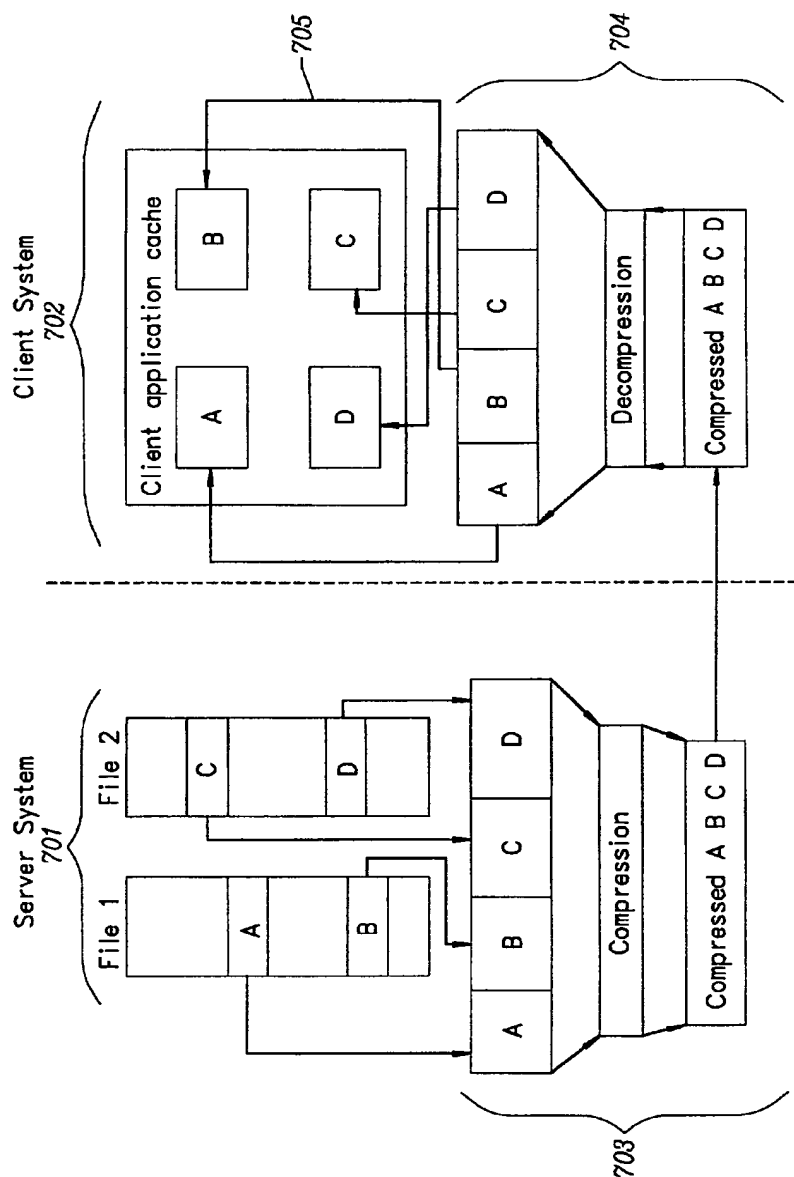


FIG. 7A

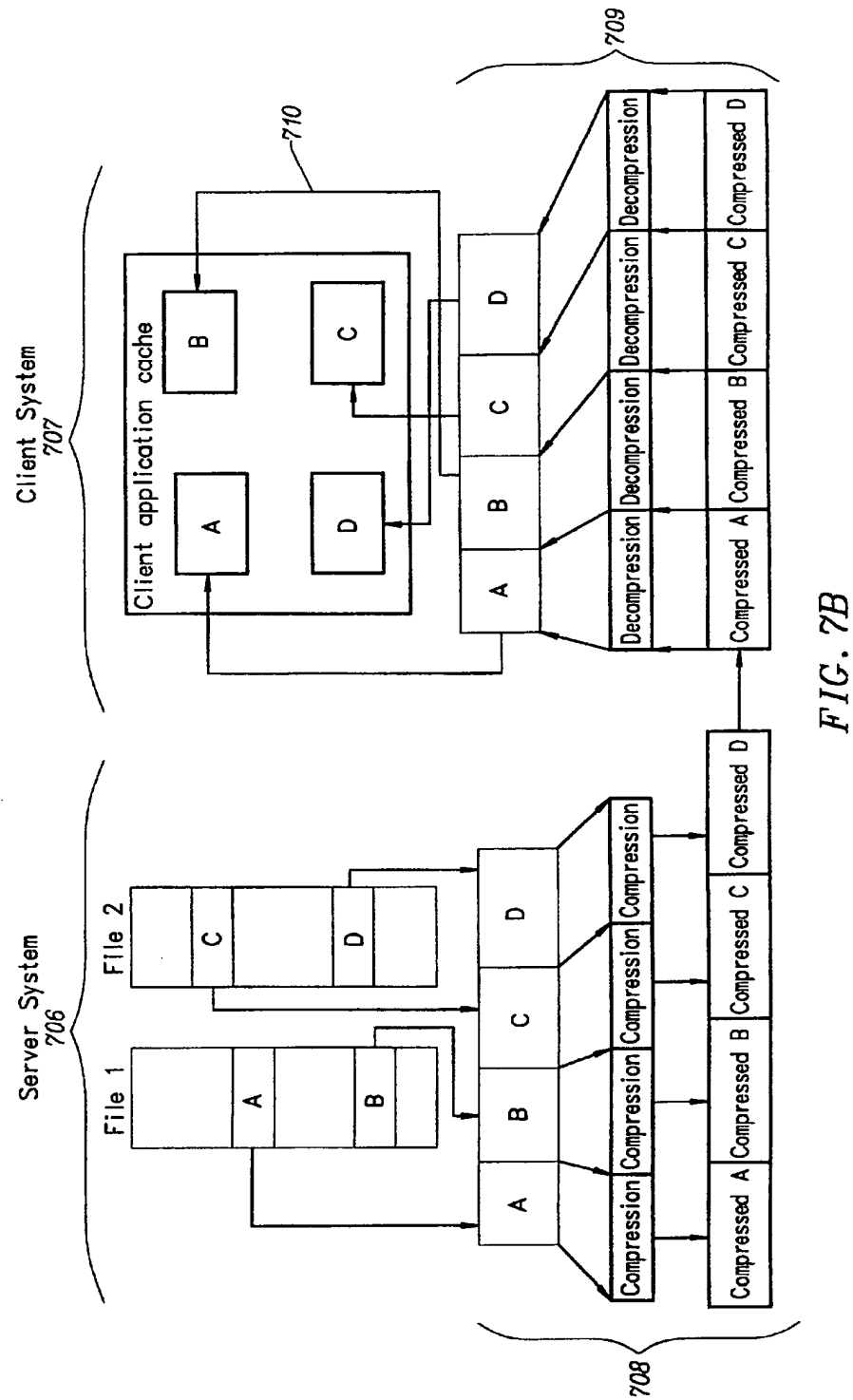


FIG. 7B

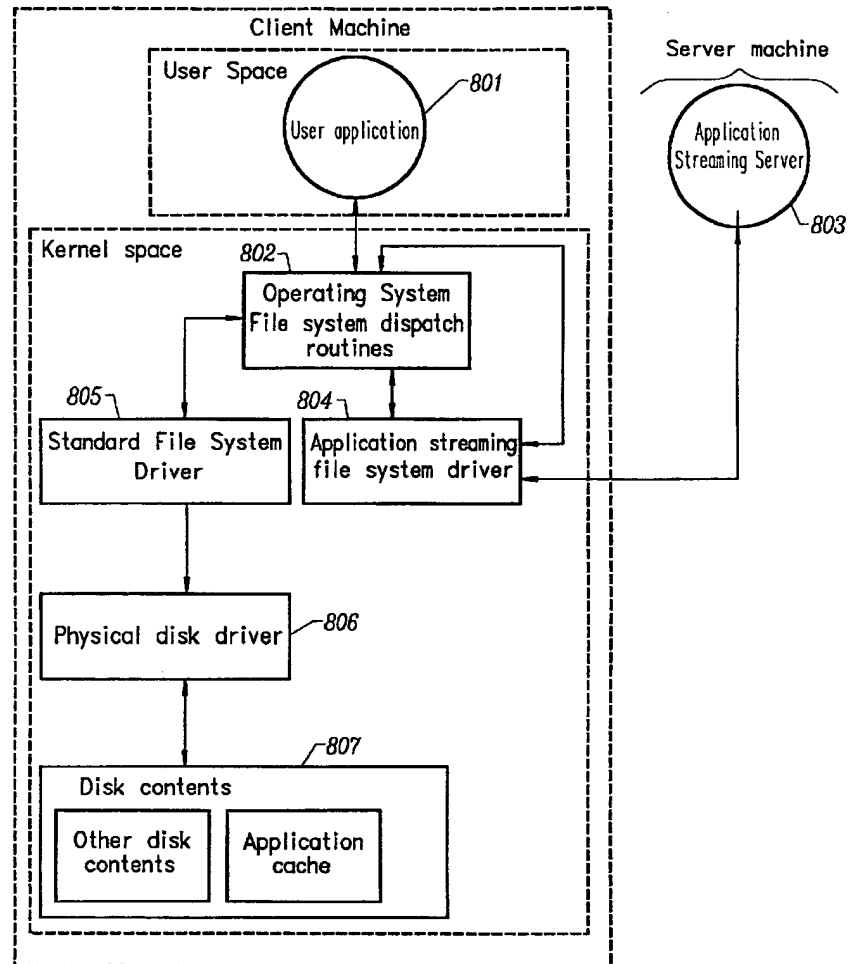


FIG. 8

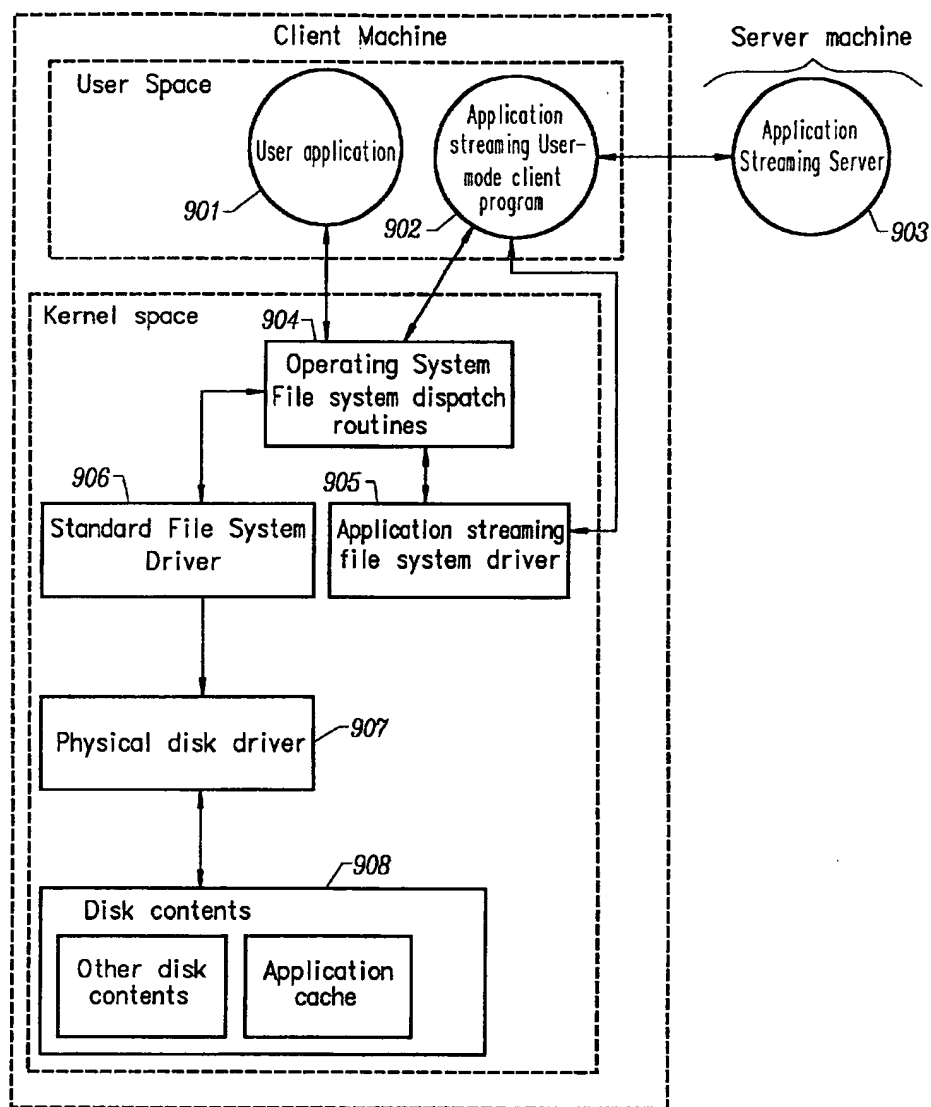


FIG. 9

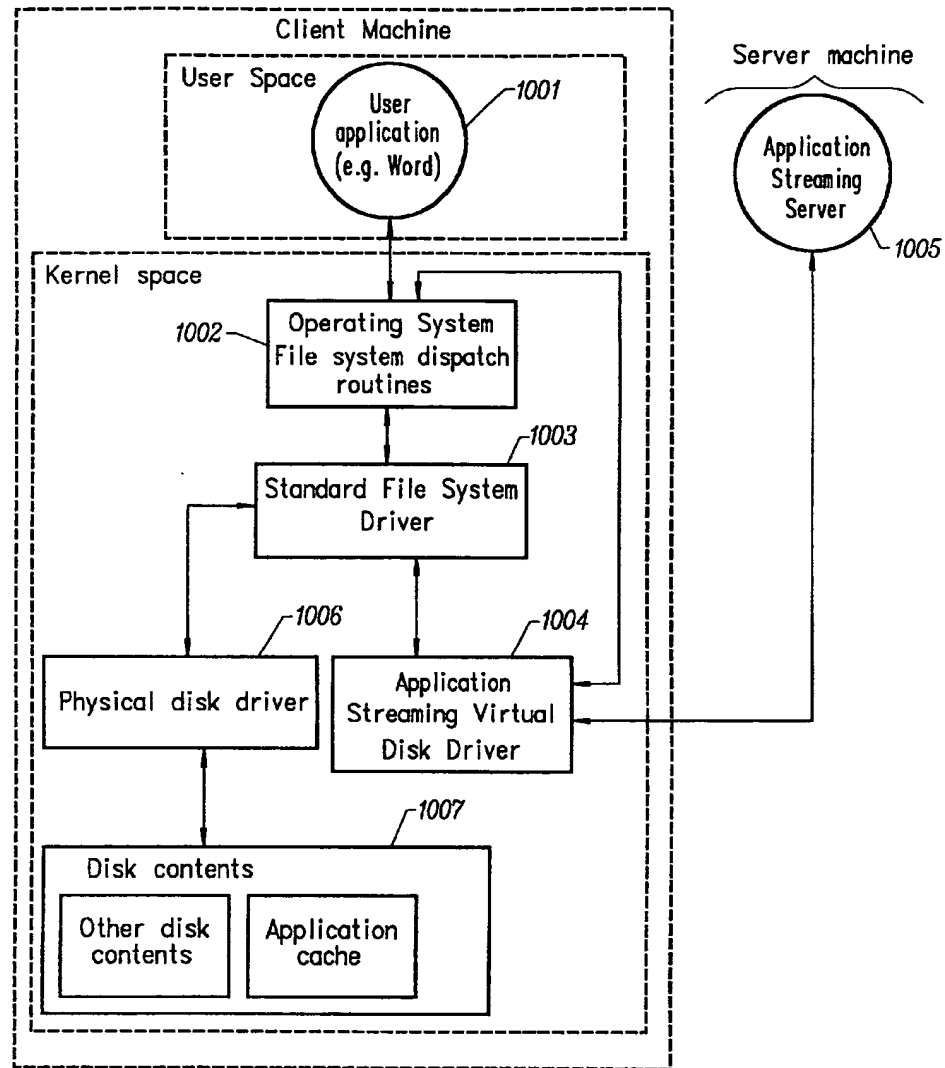


FIG. 10

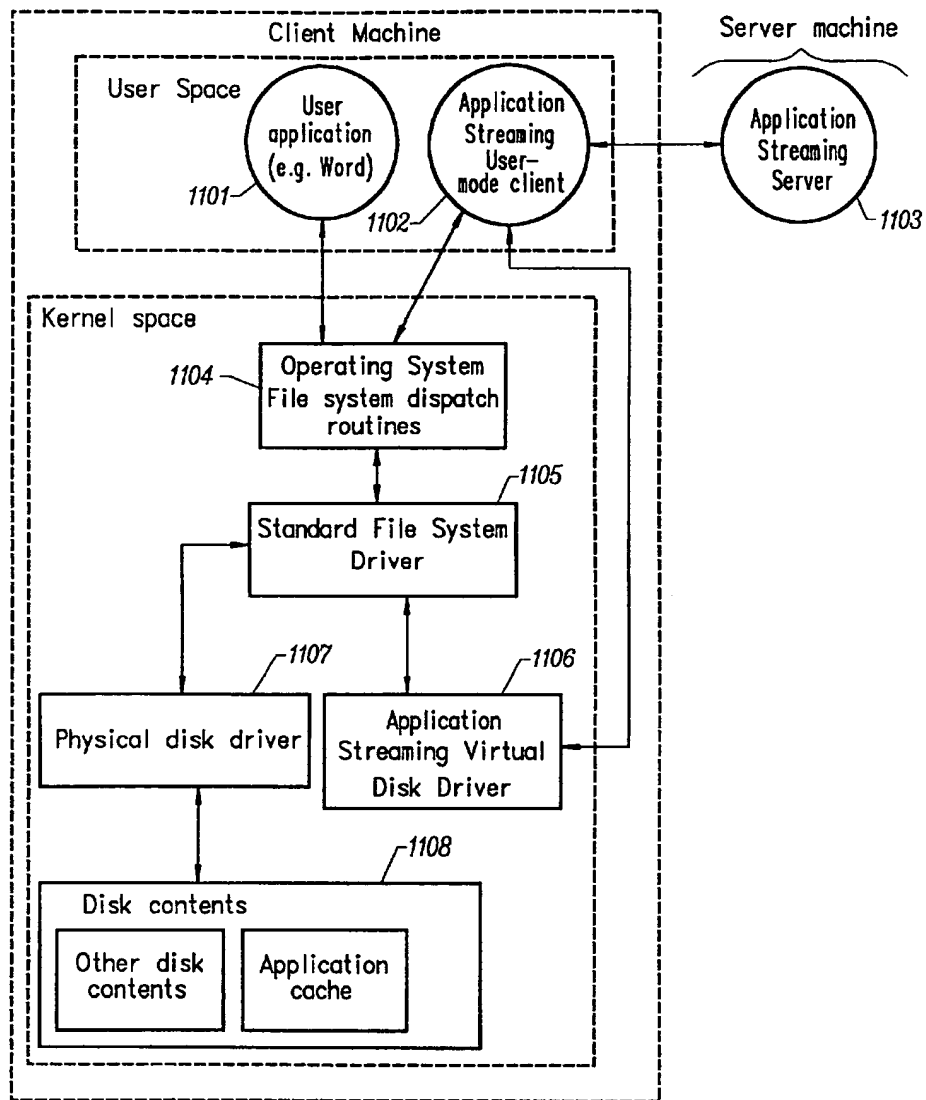


FIG. 11

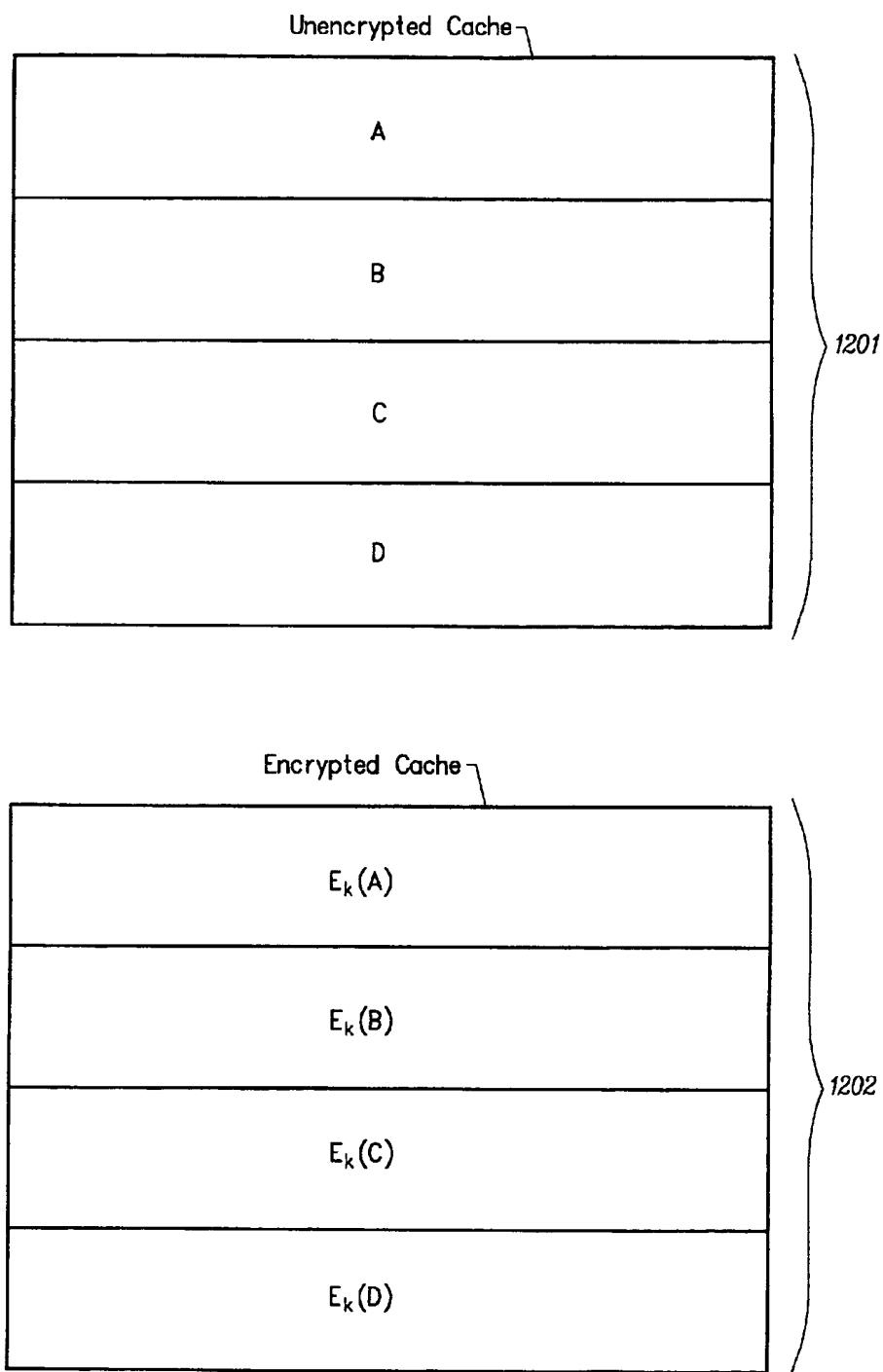


FIG. 12

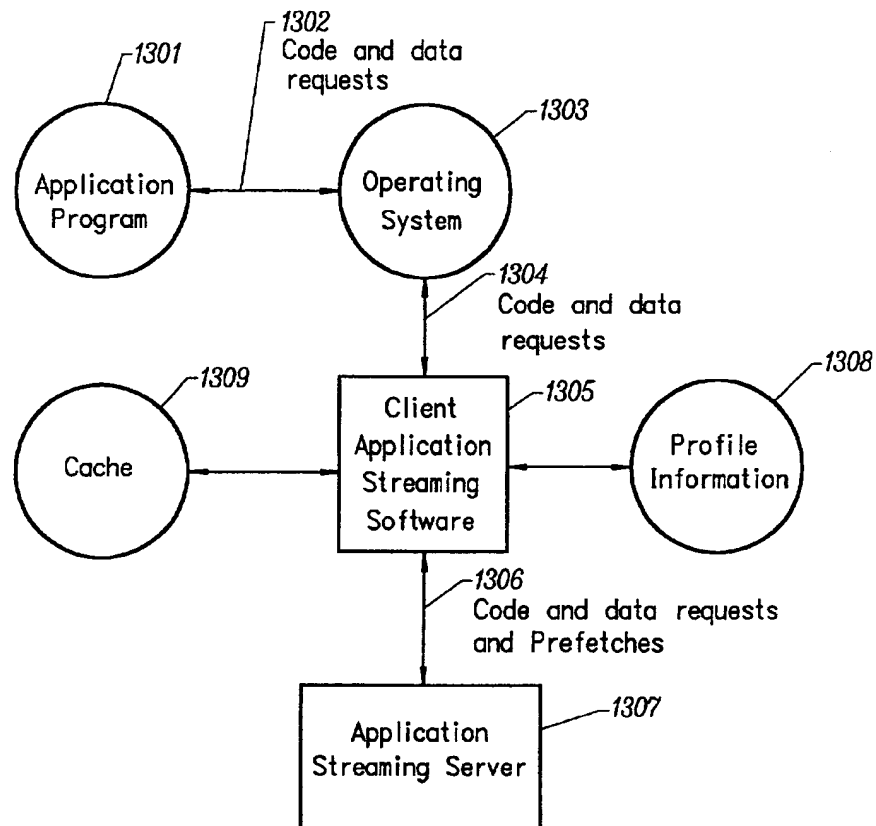


FIG. 13

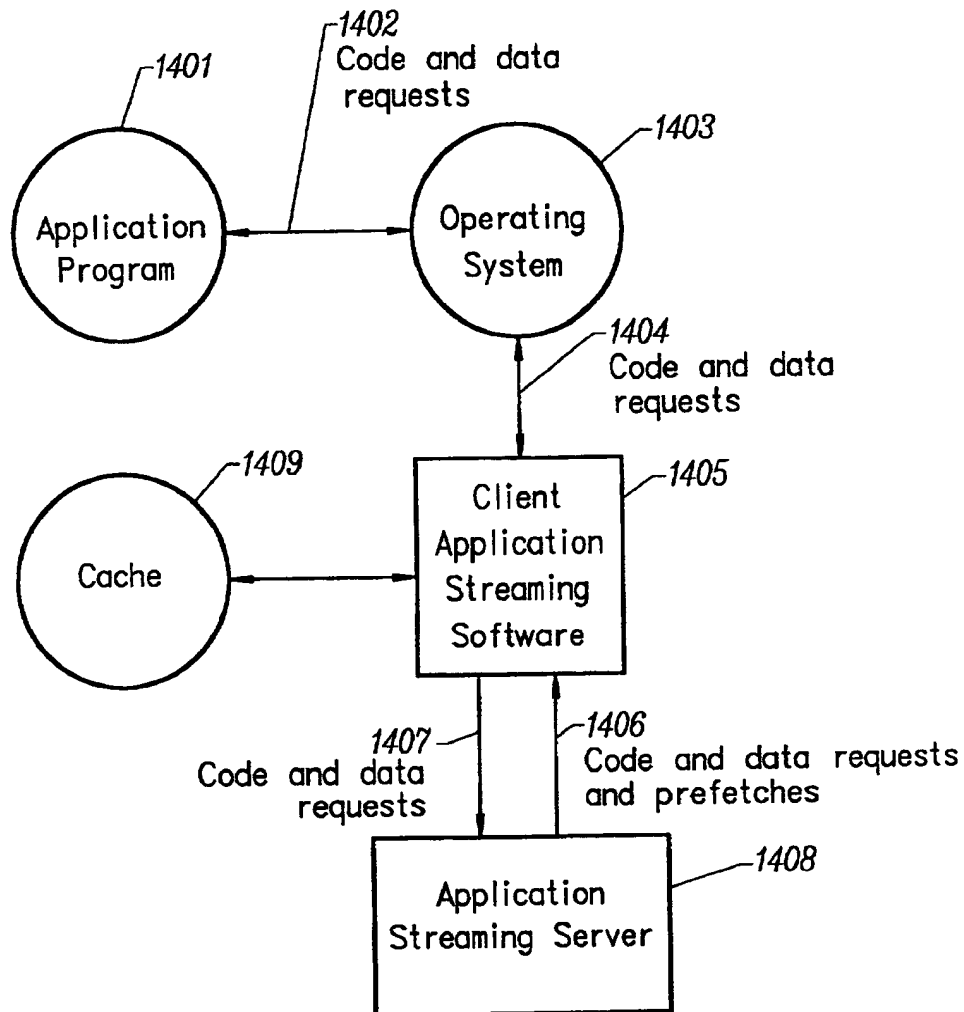
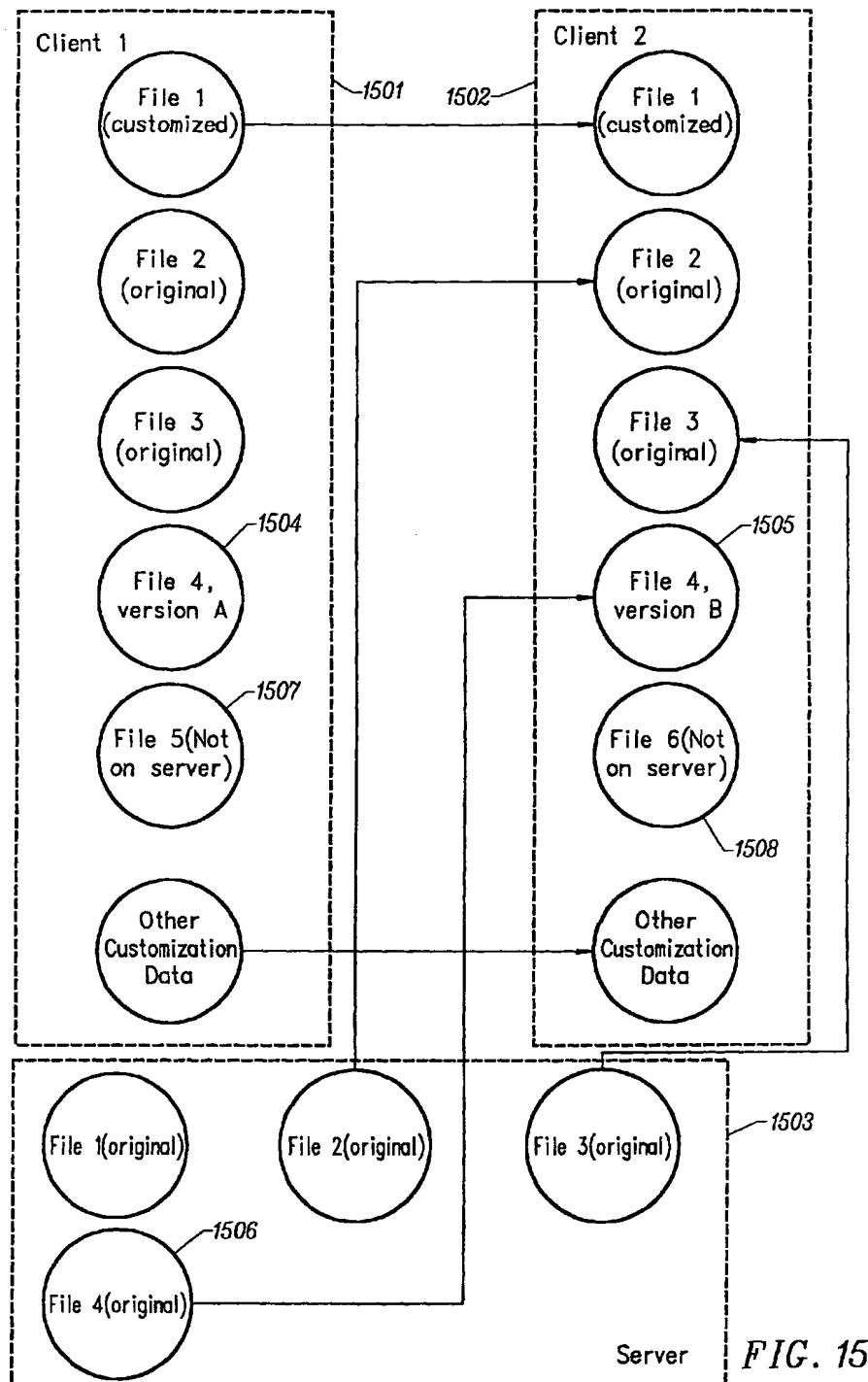


FIG. 14



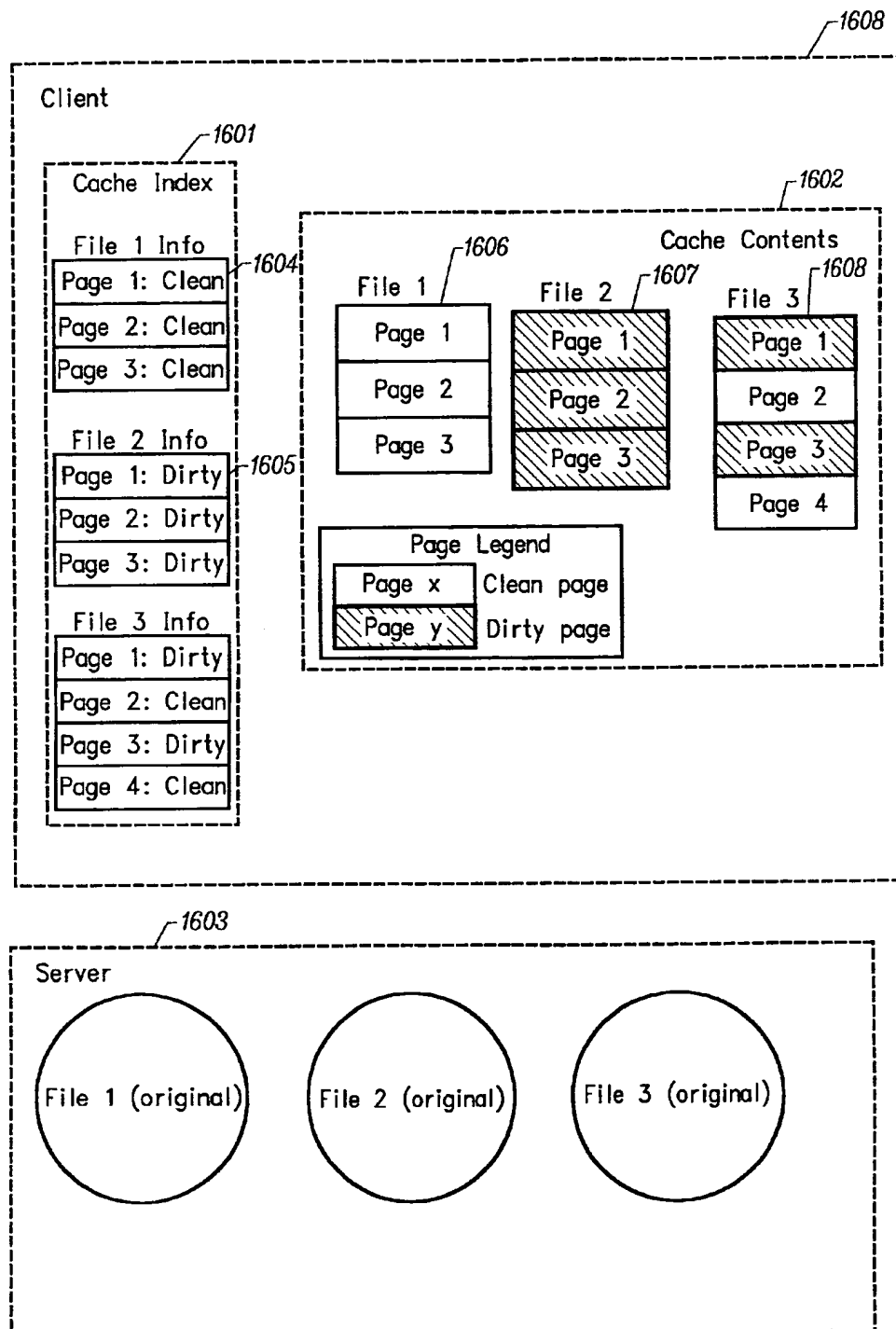


FIG. 16

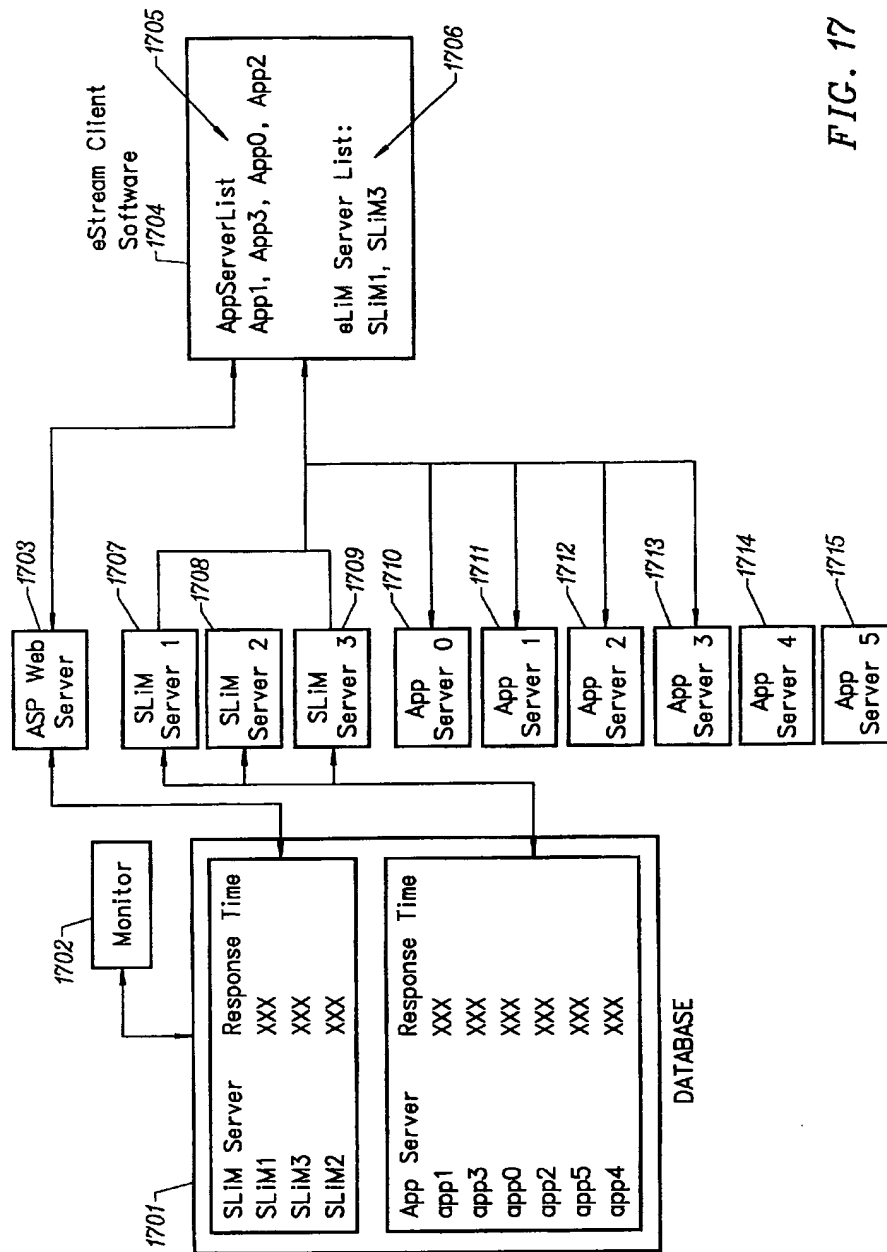
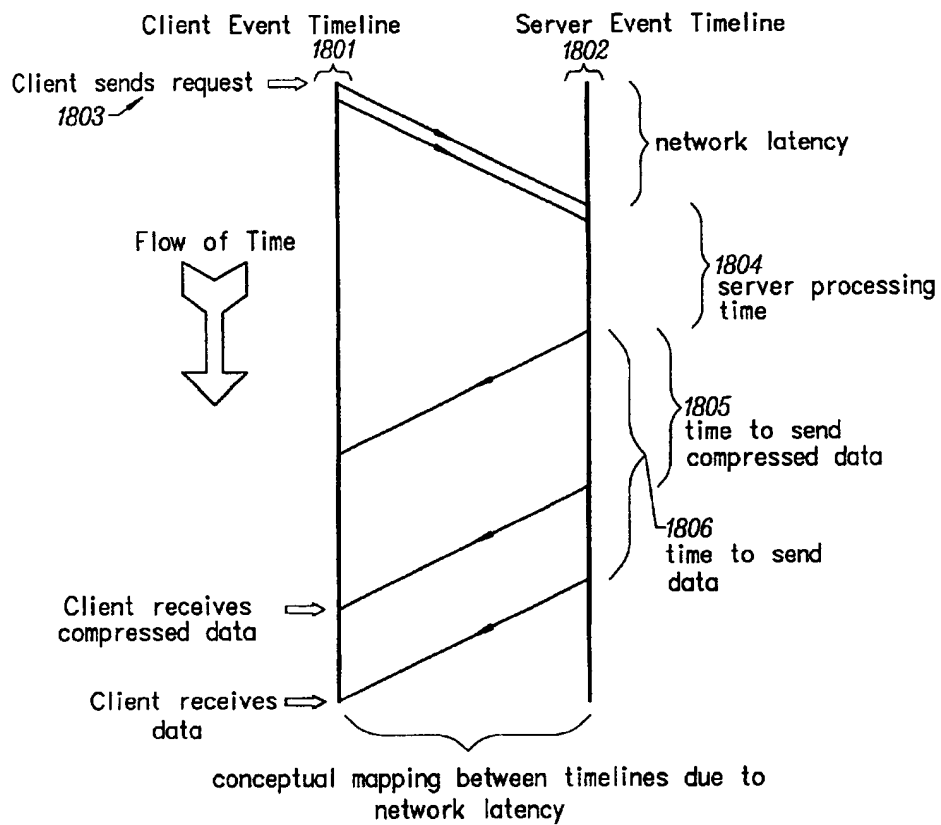


FIG. 17



Client receives data sooner if it is compressed

FIG. 18

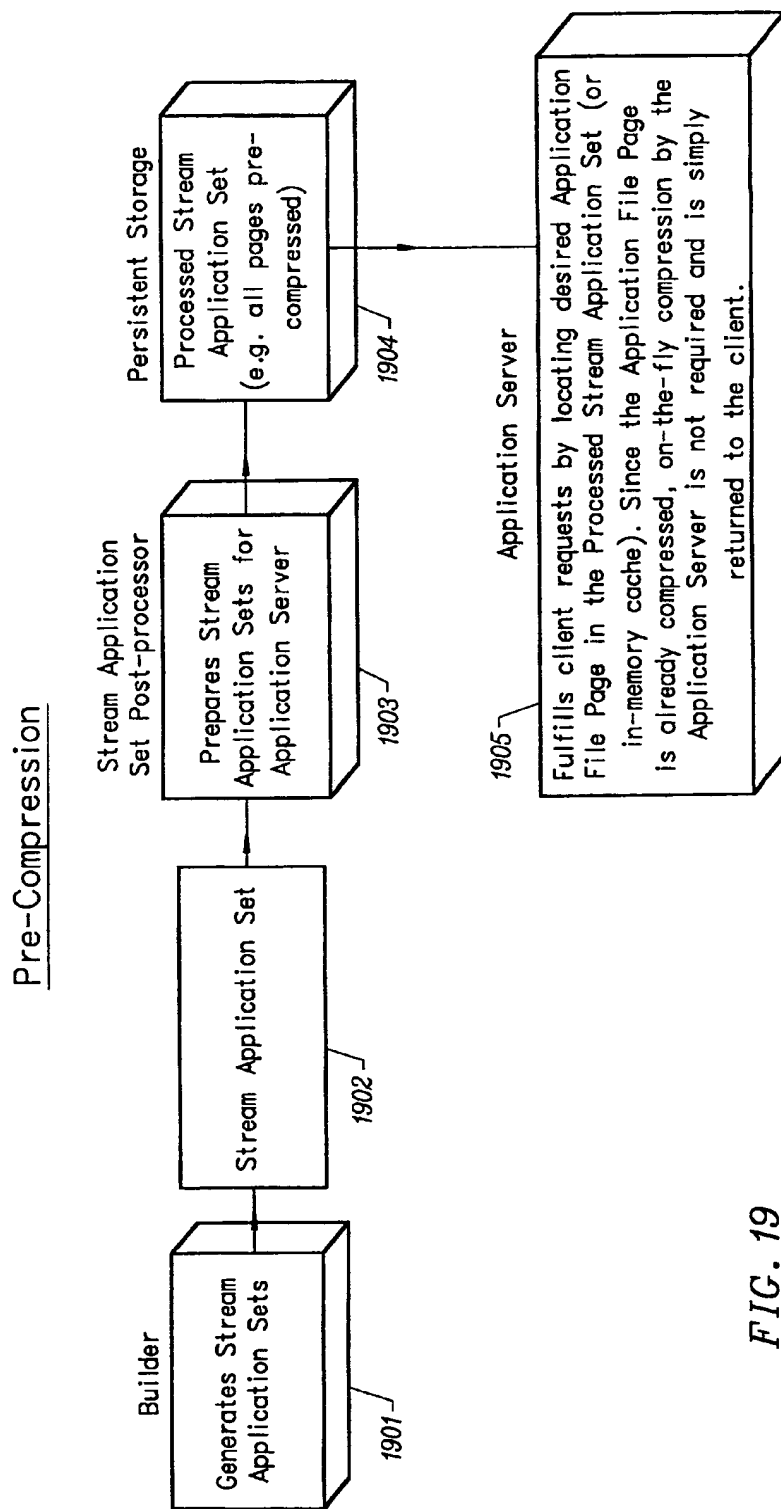
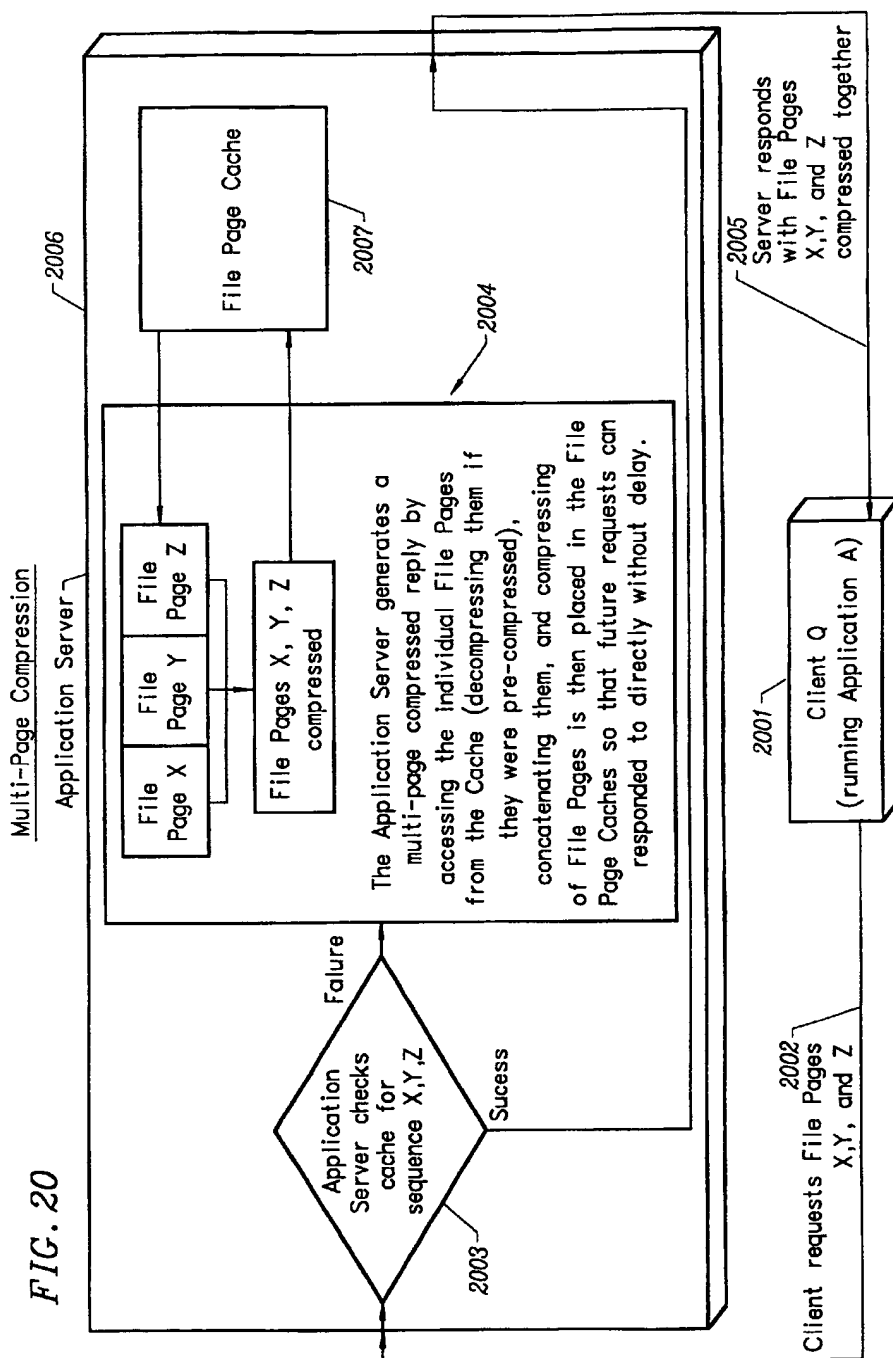
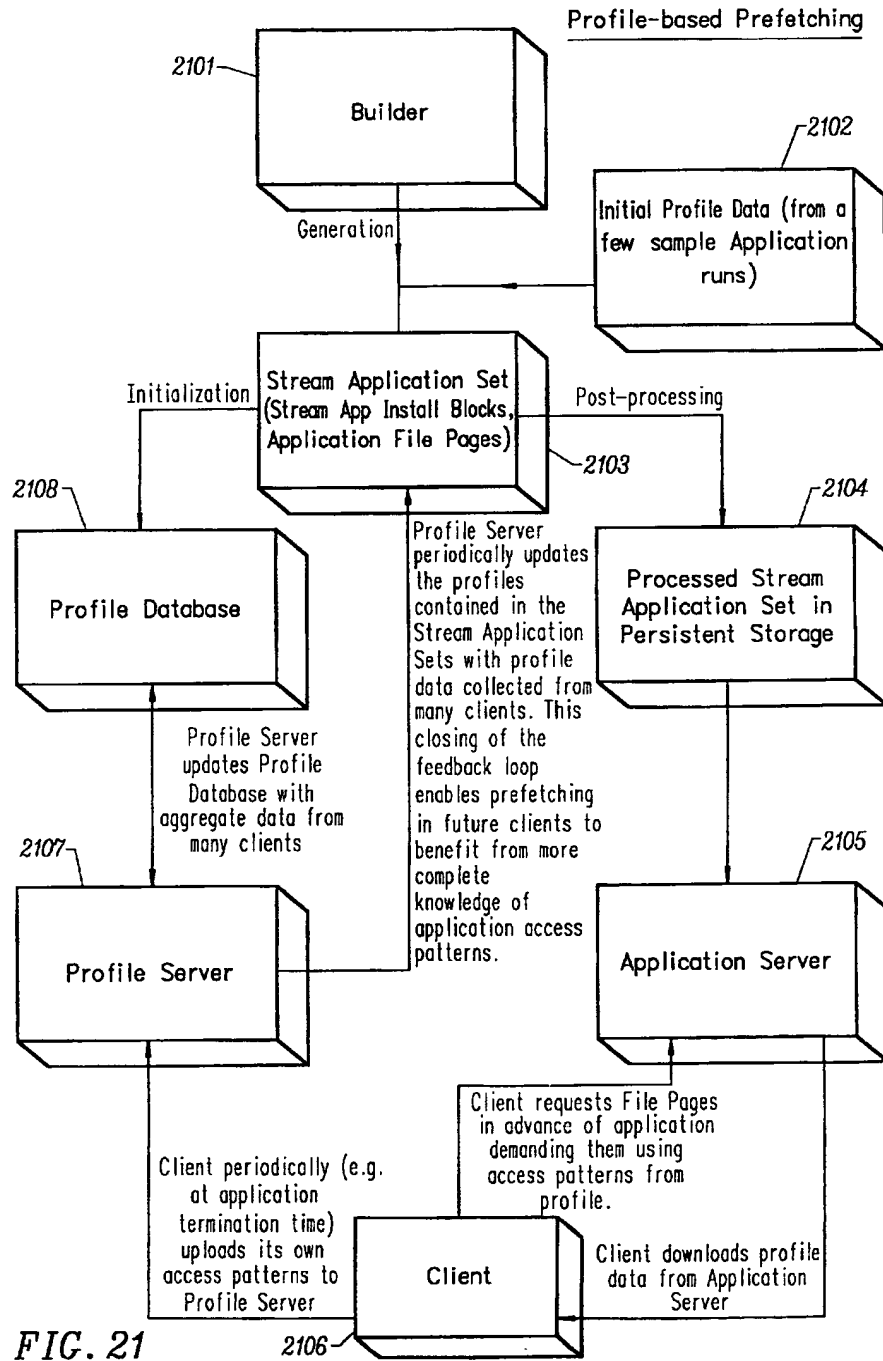


FIG. 19





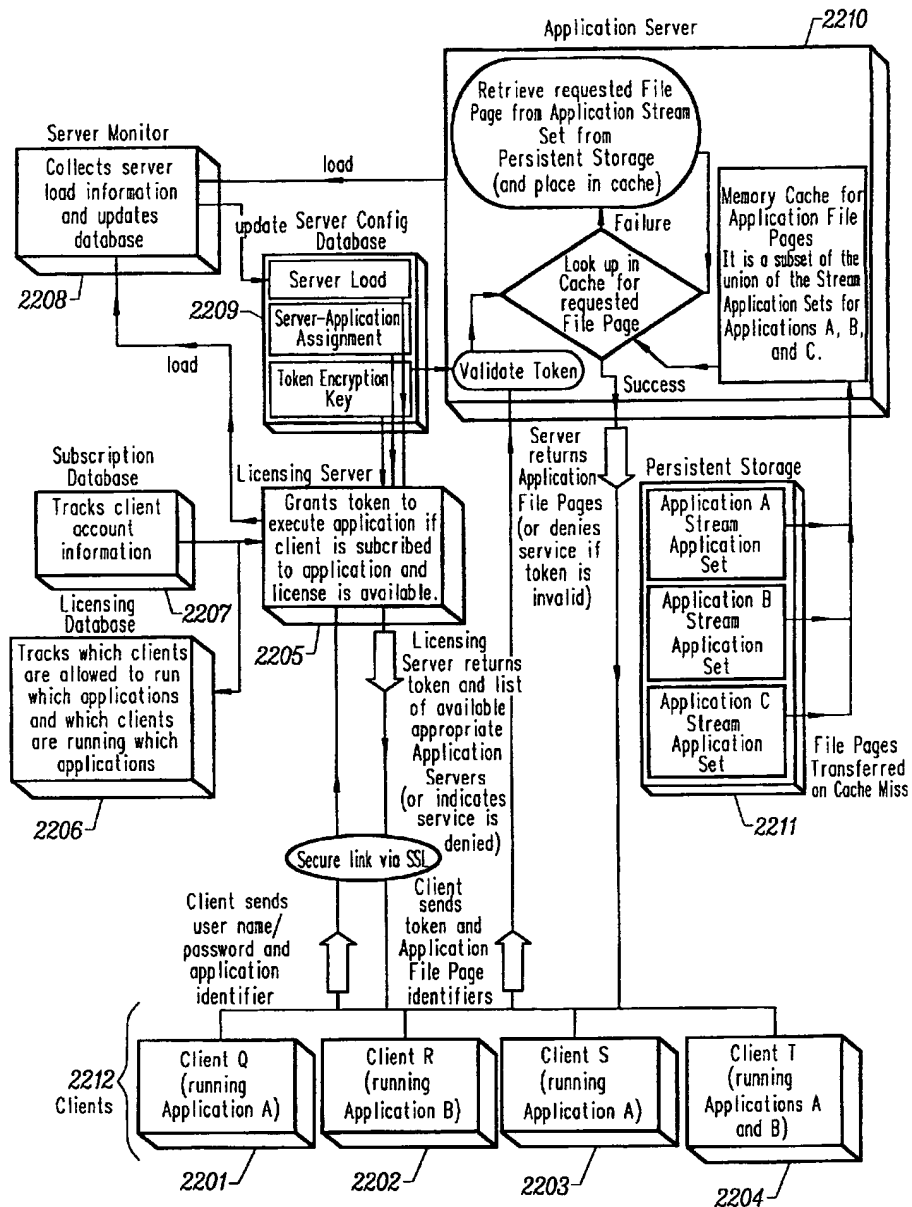


FIG. 22

Builder Install Monitor (IM) Control Flow Diagram

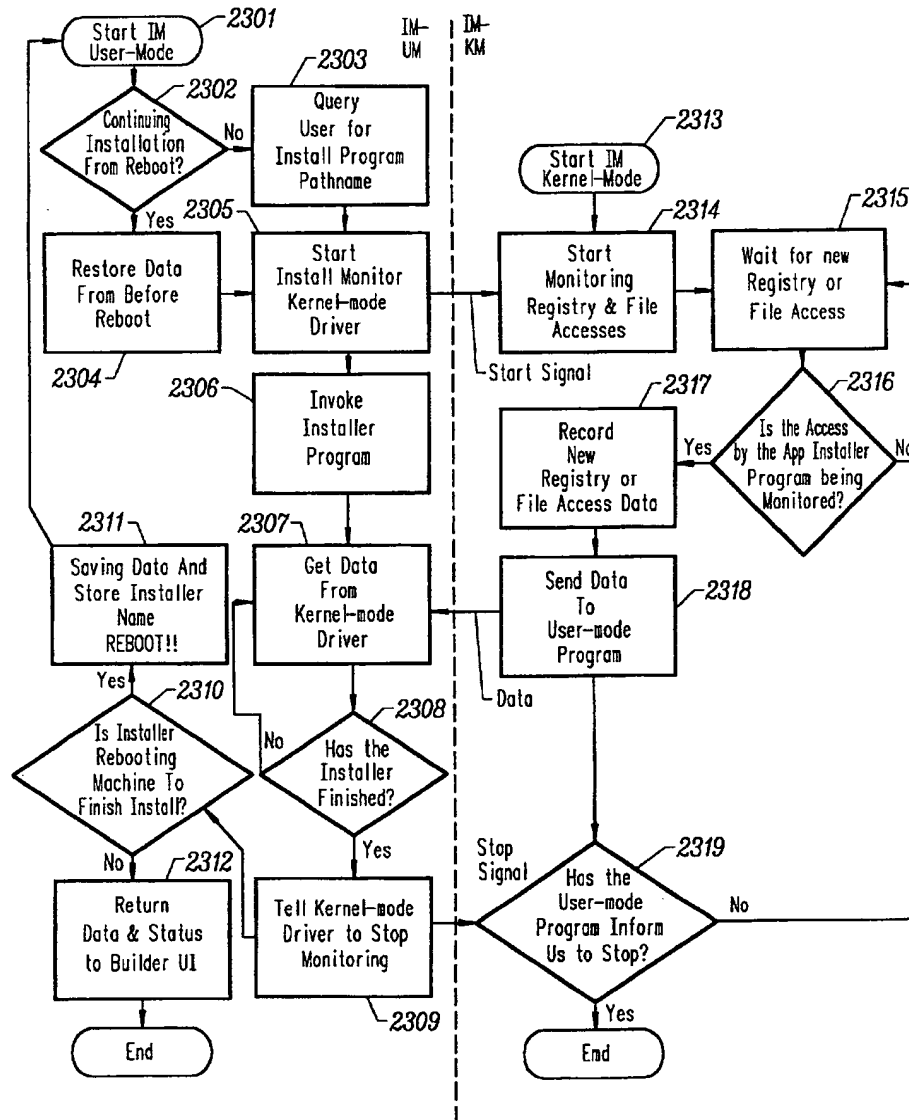


FIG. 23

Builder Application Profiler (AP) Control Flow Diagram

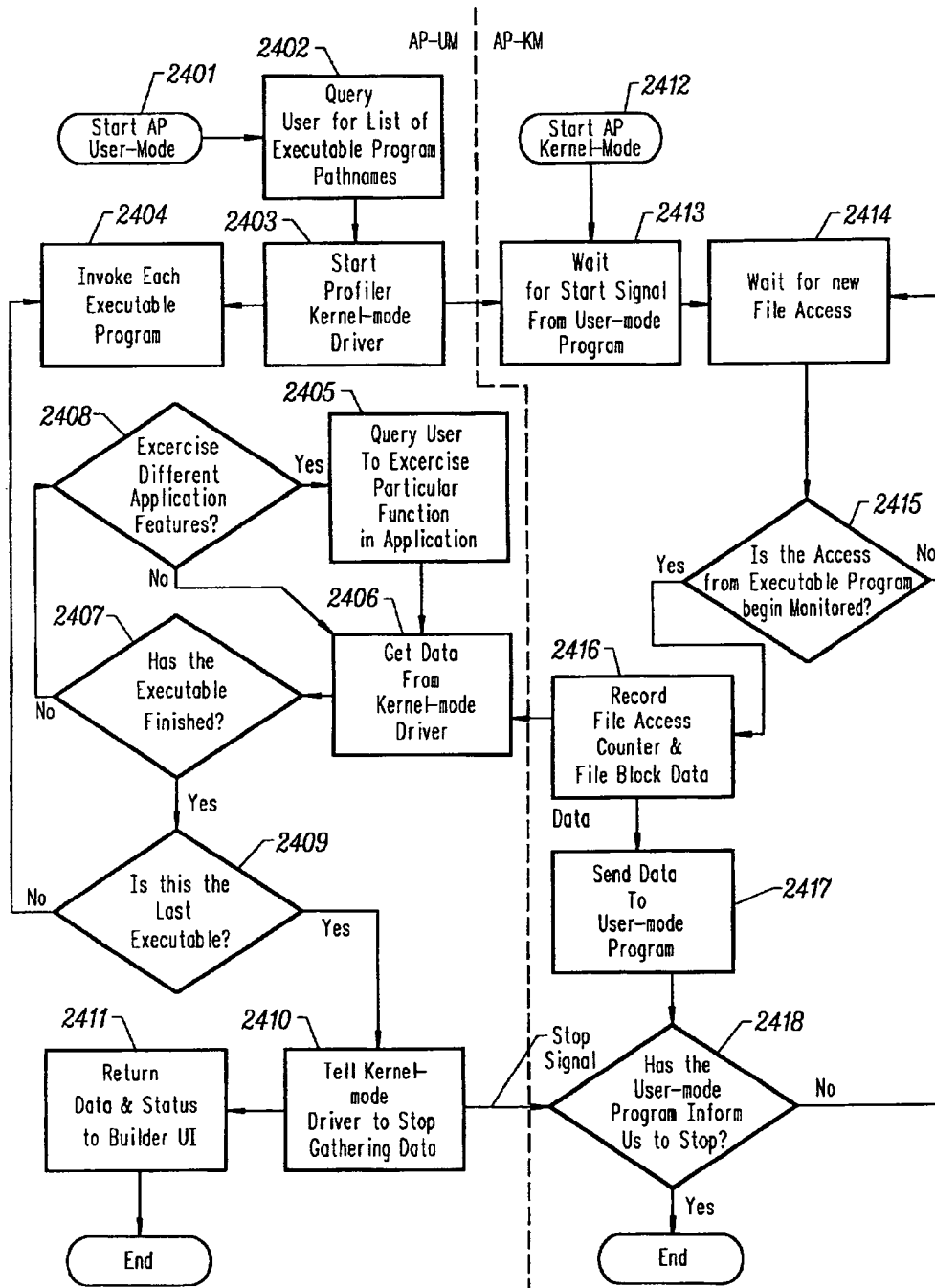


FIG. 24

Builder SAS Packager (SP) Control Flow Diagram

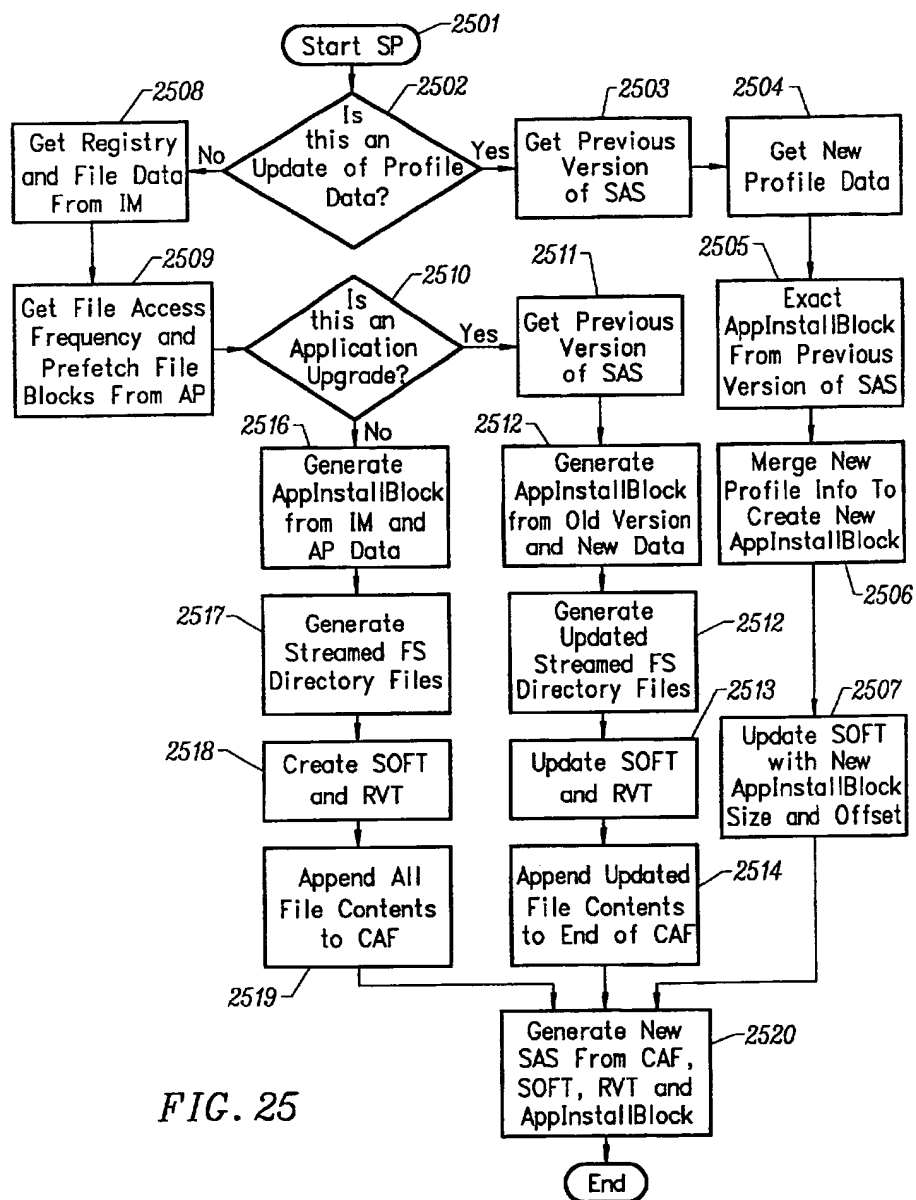


FIG. 25

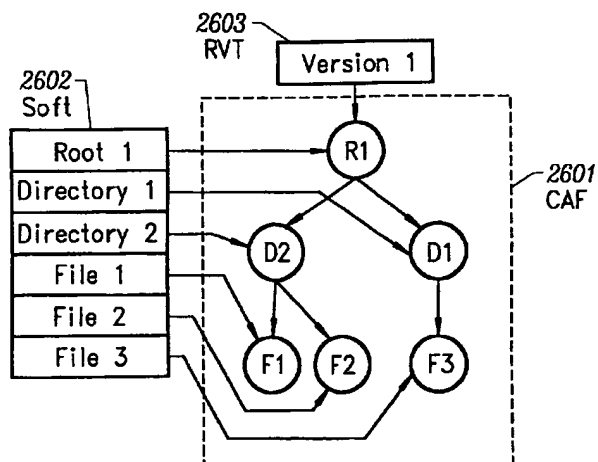


FIG. 26A

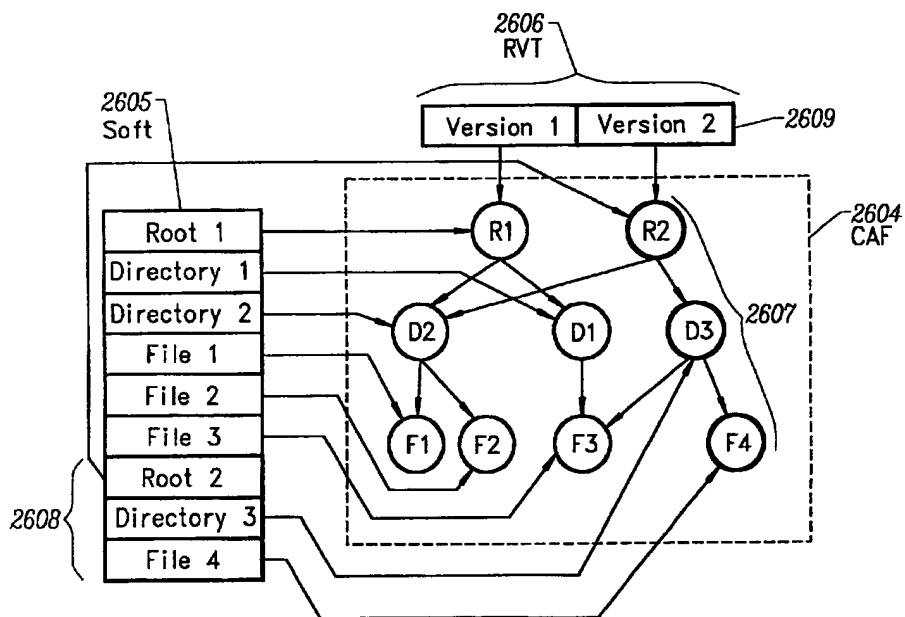
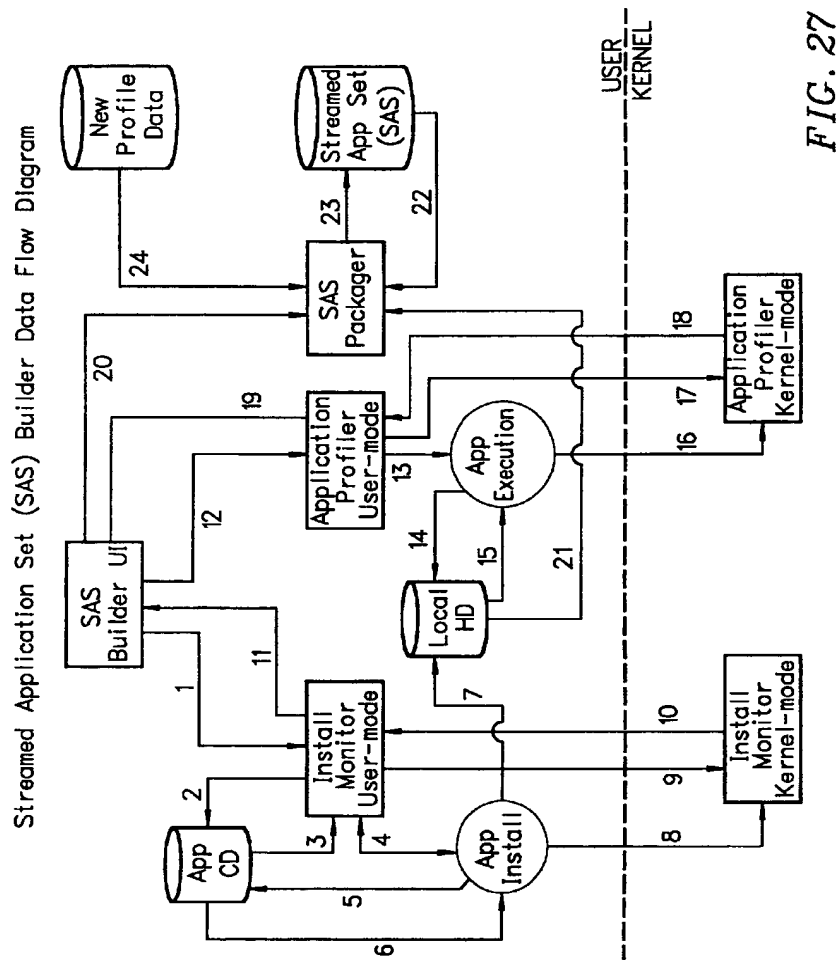


FIG. 26B



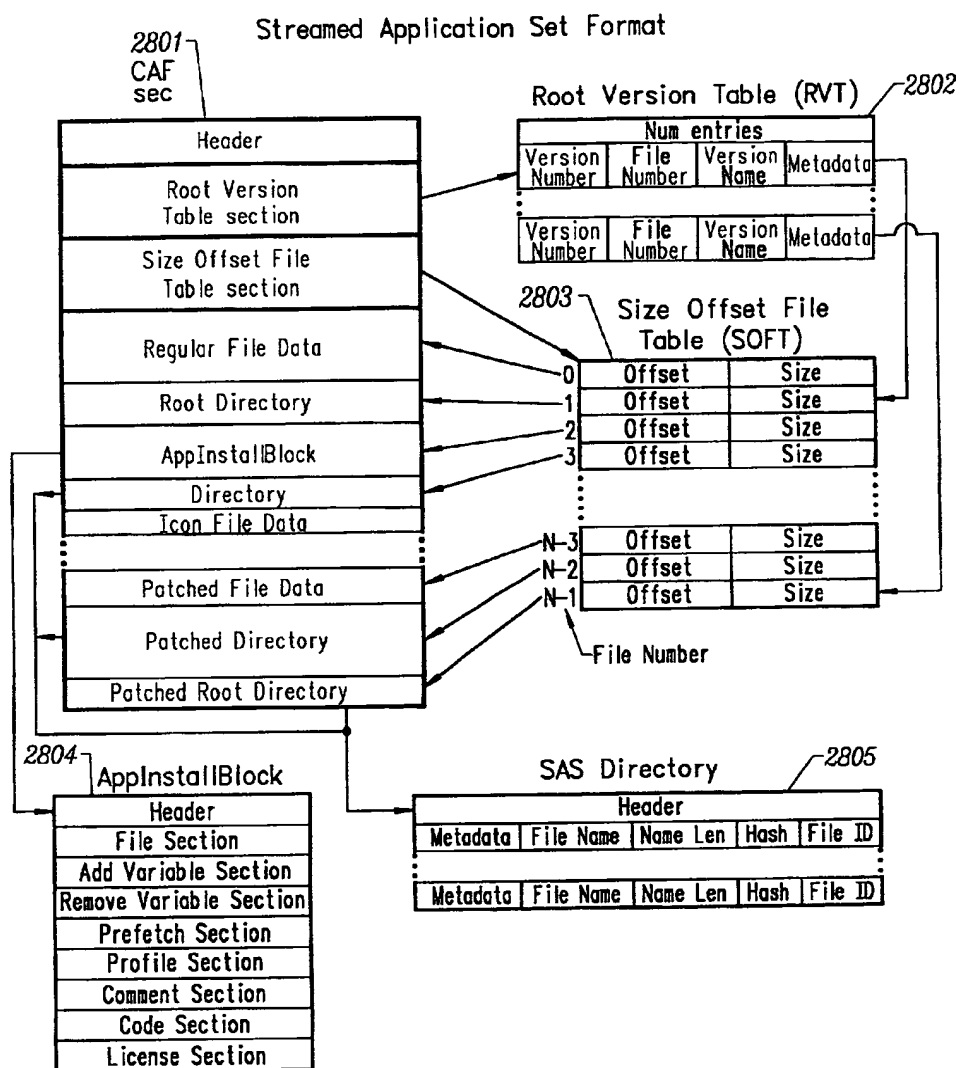


FIG. 28

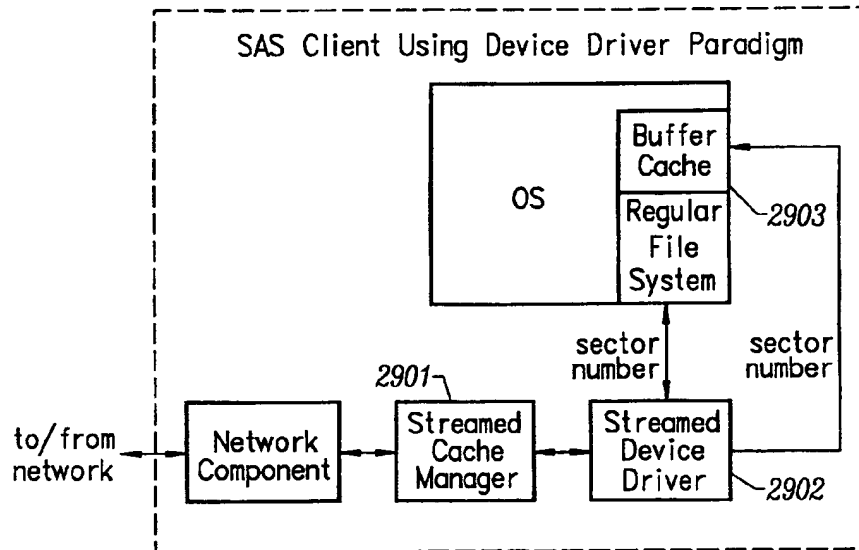


FIG. 29

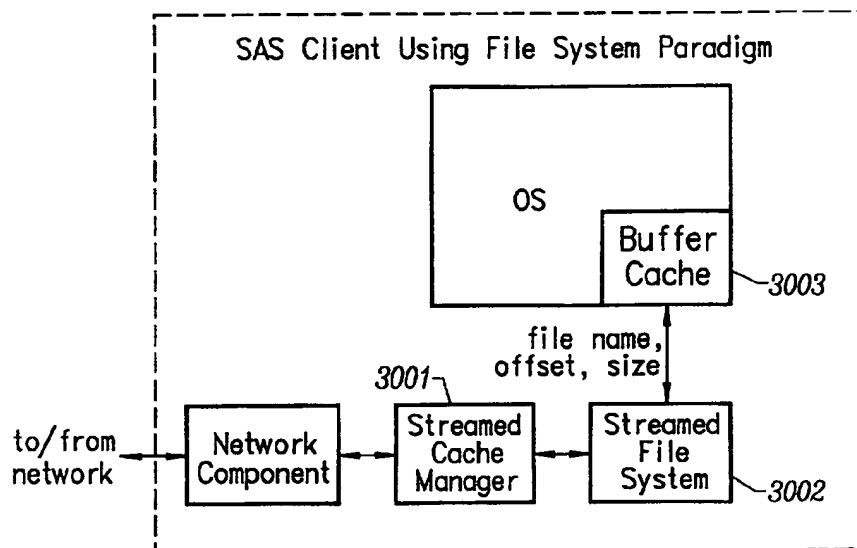


FIG. 30

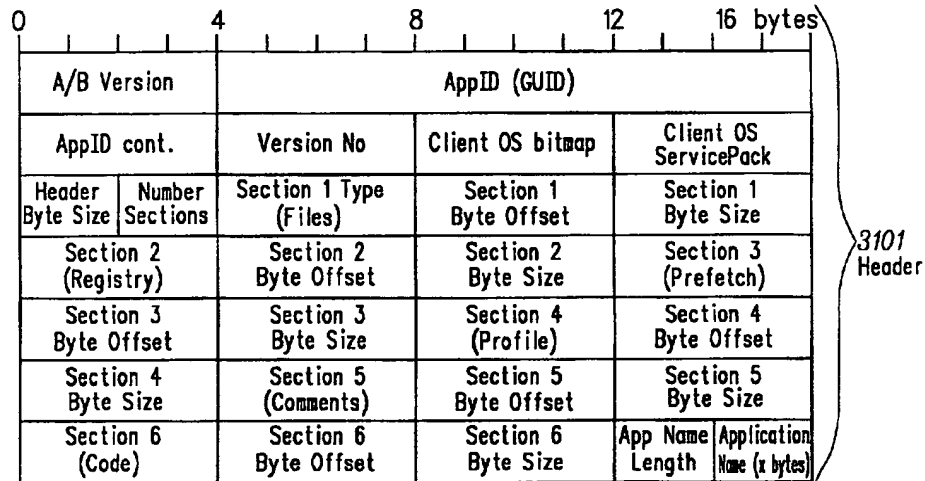


FIG. 31A

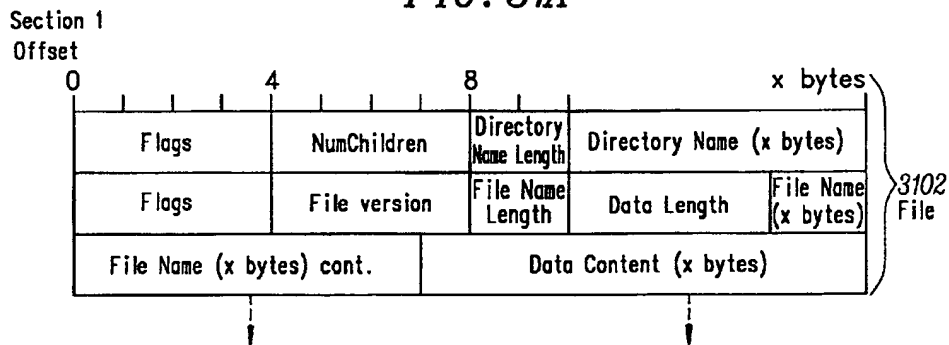


FIG. 31B

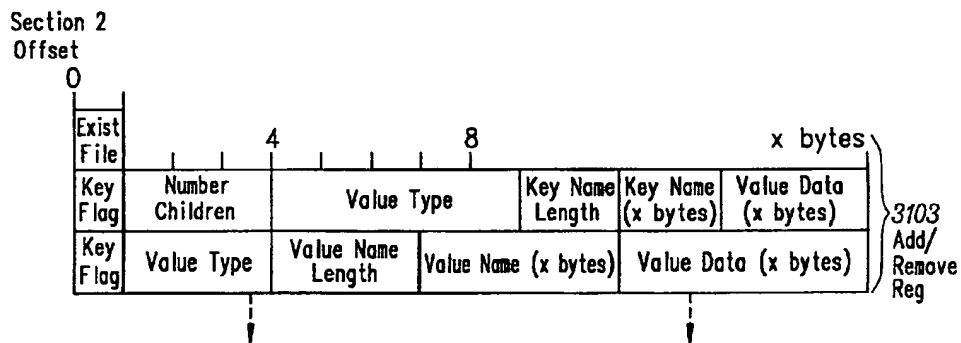


FIG. 31C

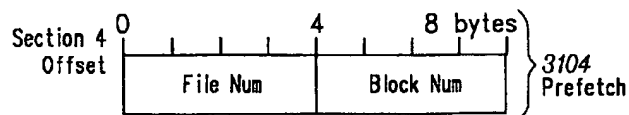


FIG. 31D

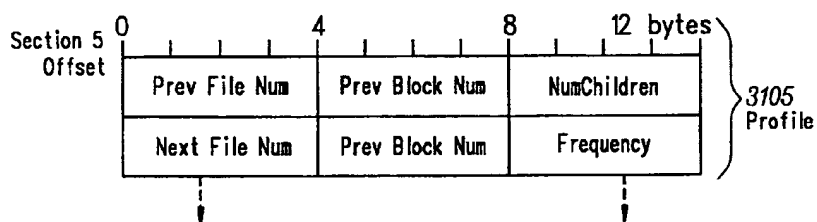


FIG. 31E

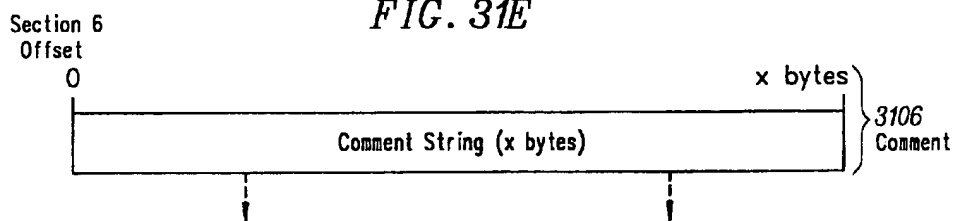


FIG. 31F

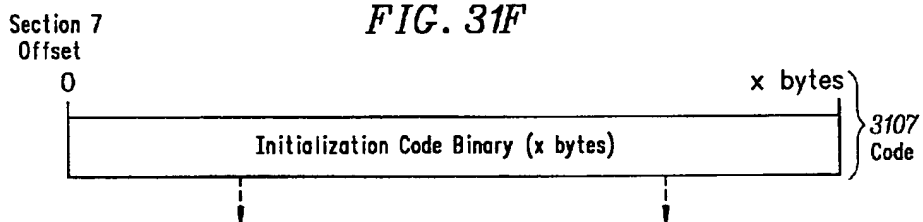


FIG. 31G

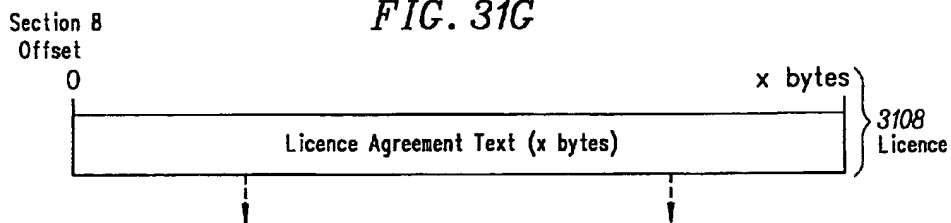


FIG. 31H

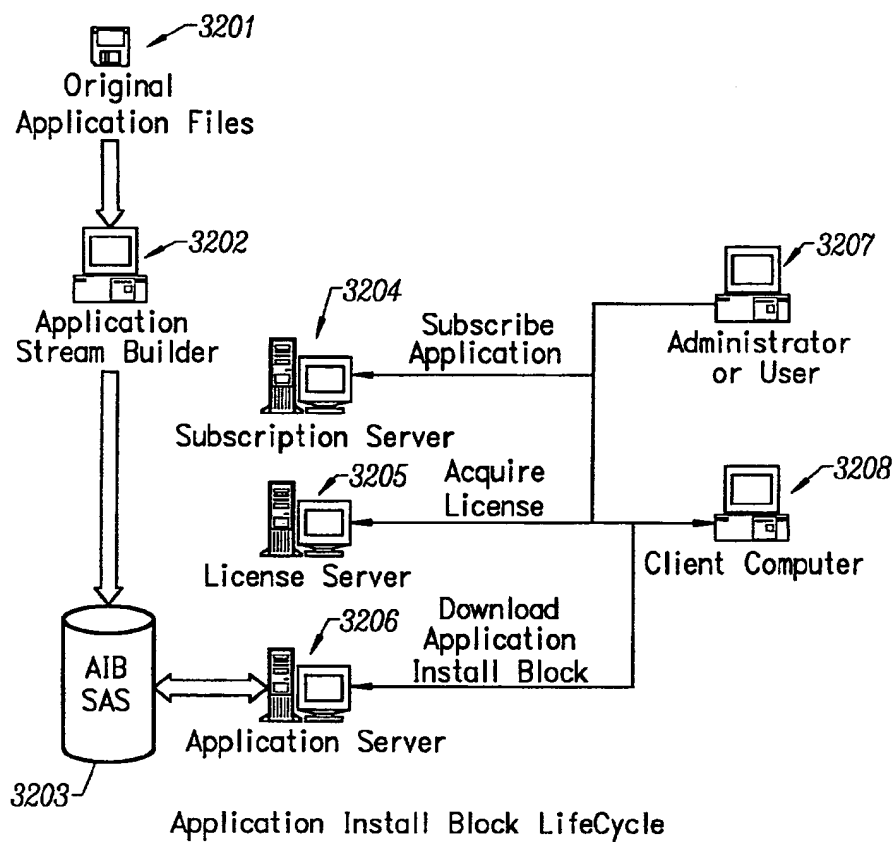


FIG. 32

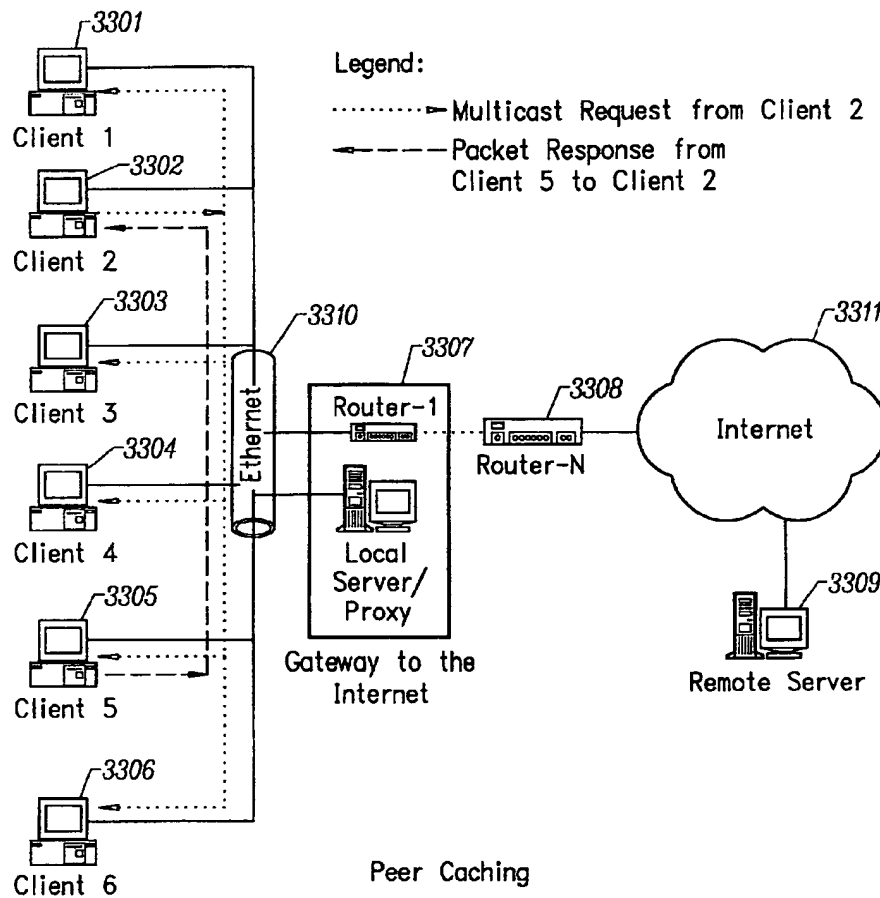


FIG. 33

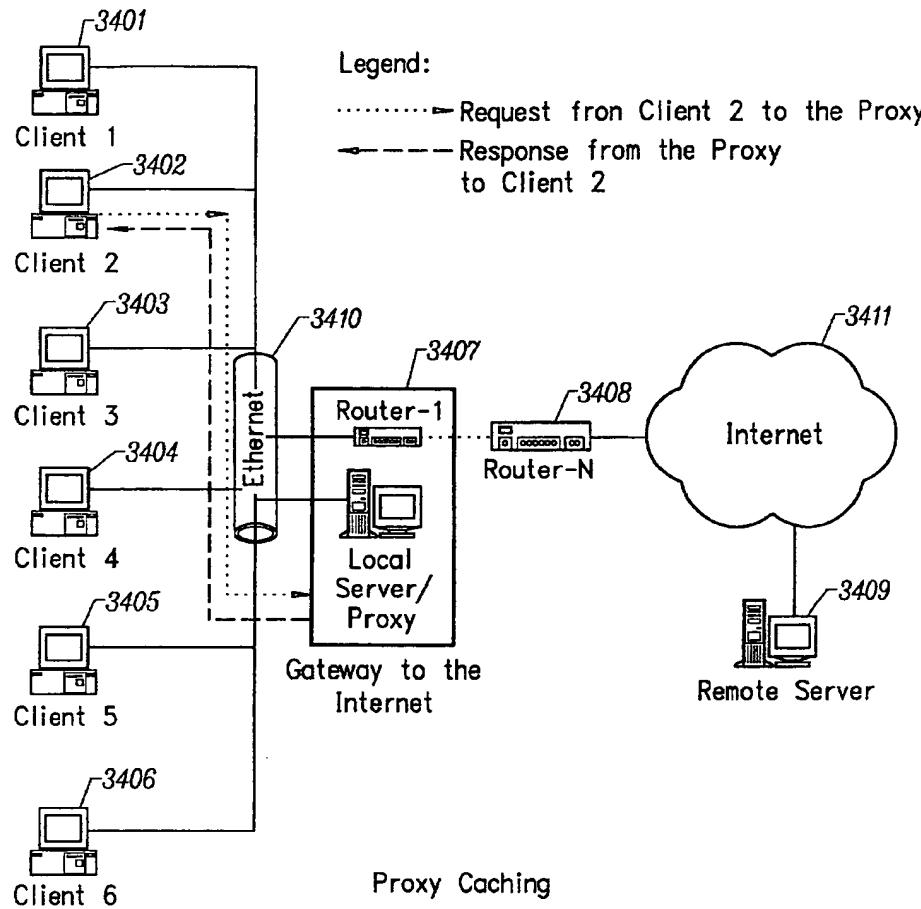


FIG. 34

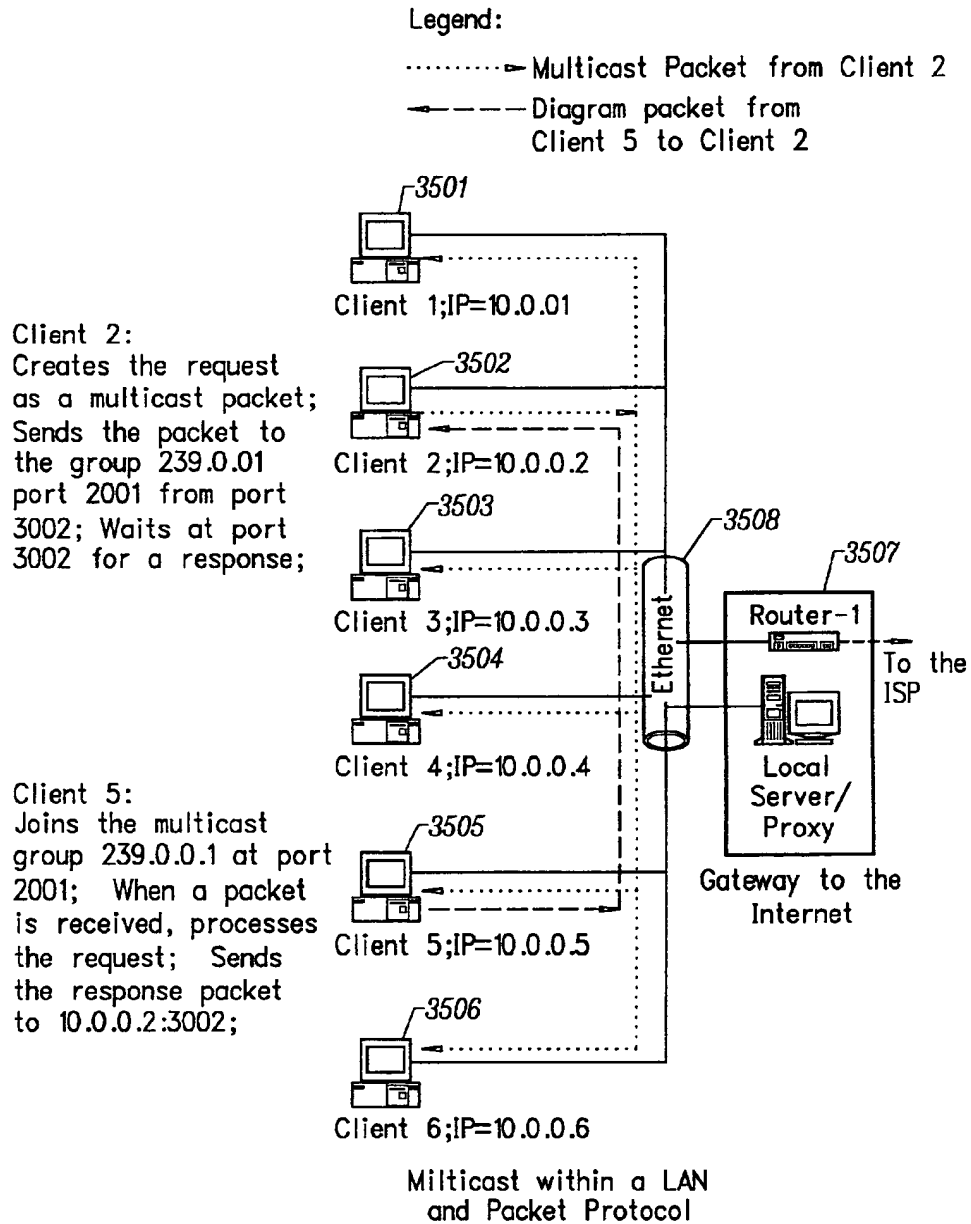
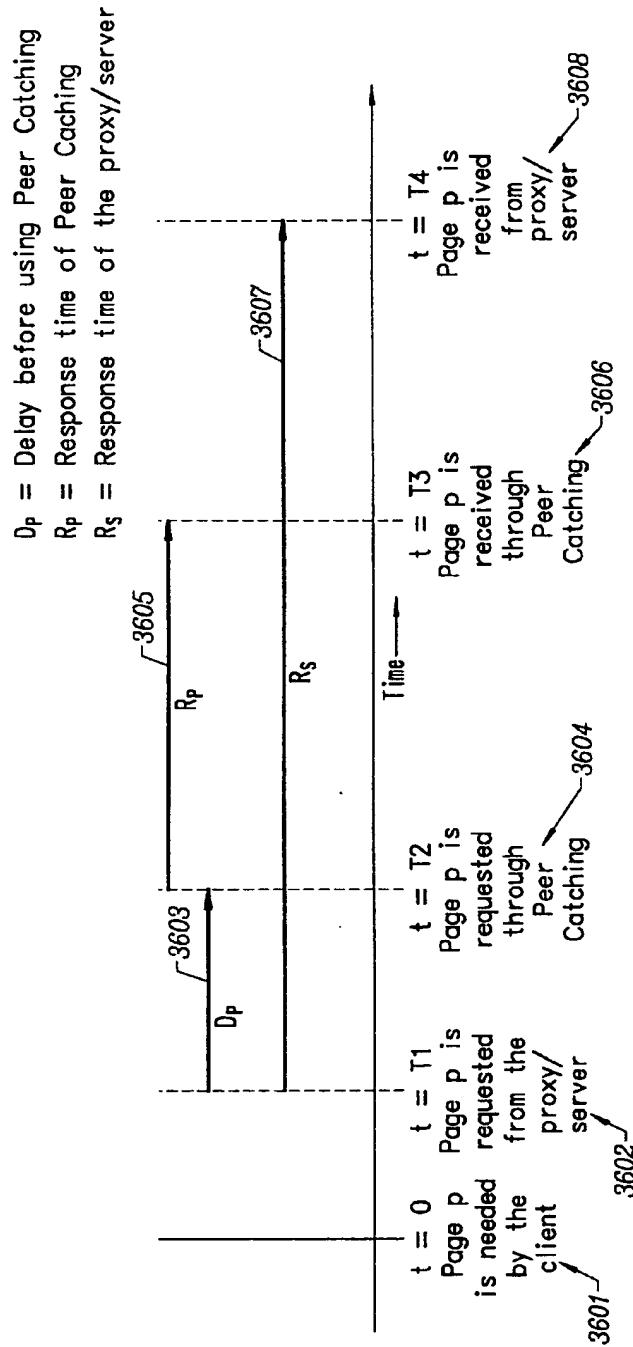
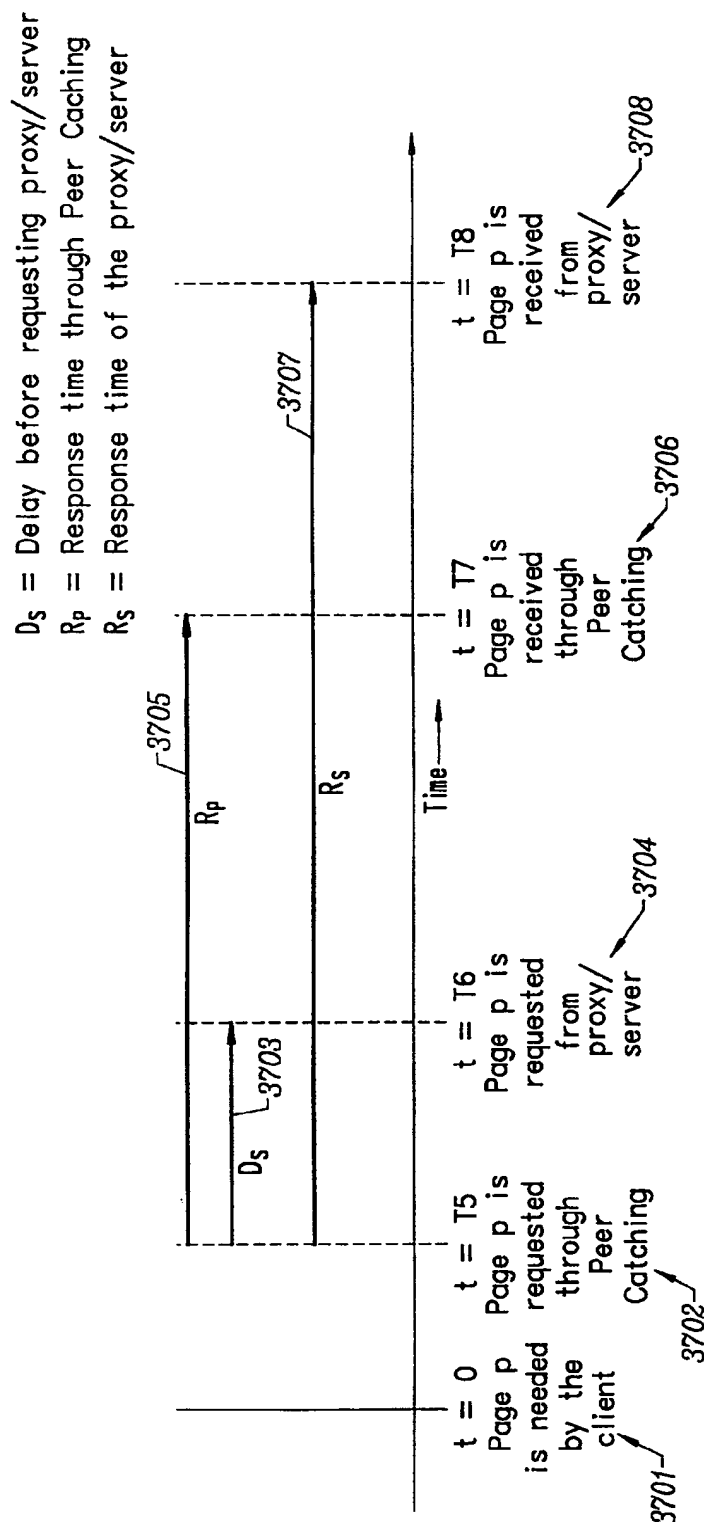


FIG. 35



Concurrent Requesting - Proxy First

FIG. 36



Concurrent Requesting - Peer Catching First

FIG. 37

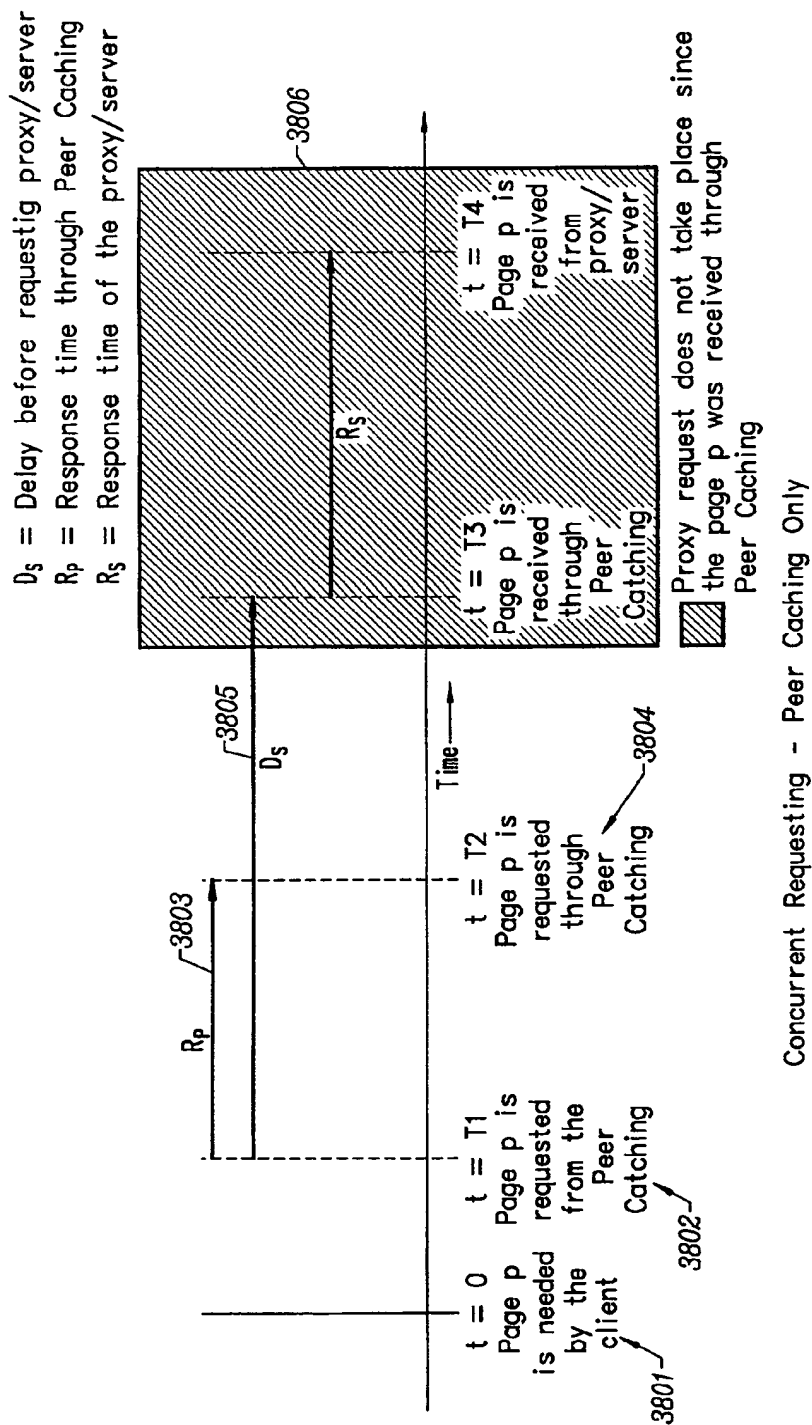
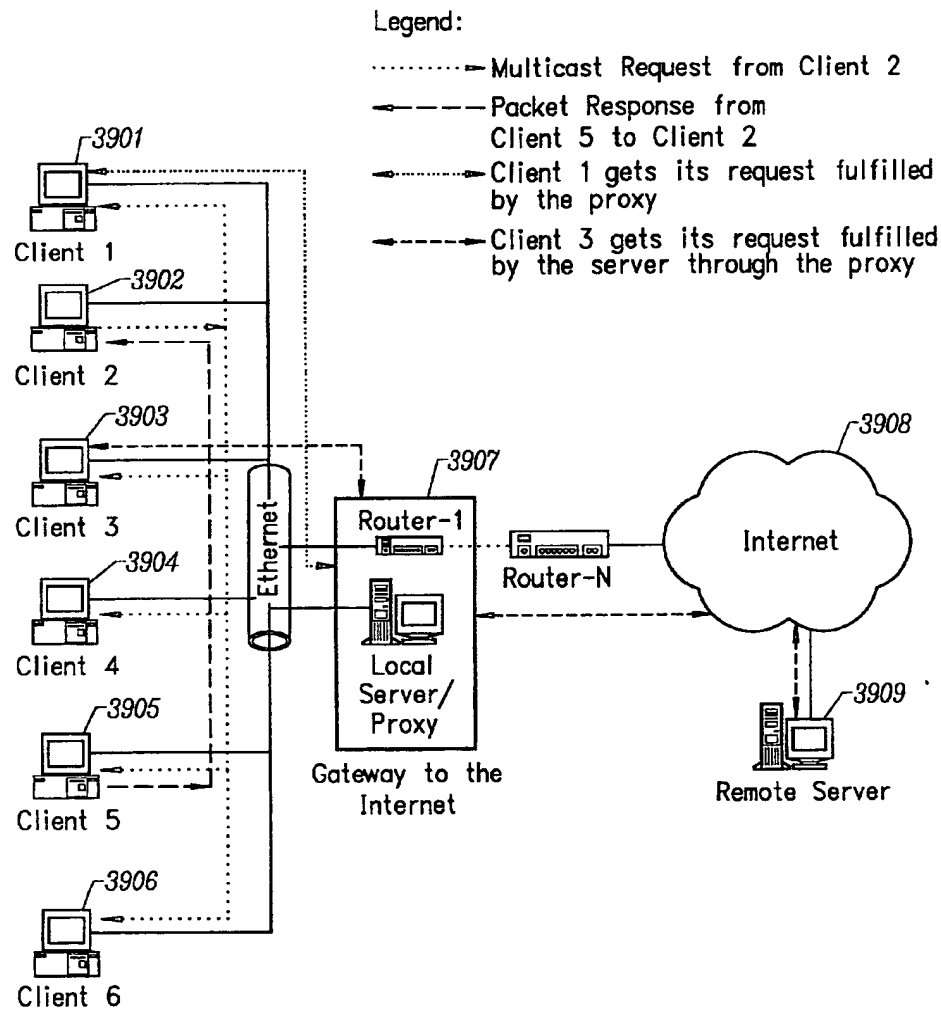


FIG. 38



Client-Server System with Peer and Proxy Caching

FIG. 39

Preventing Piracy of Remotely Served,
Locally Executed Applications

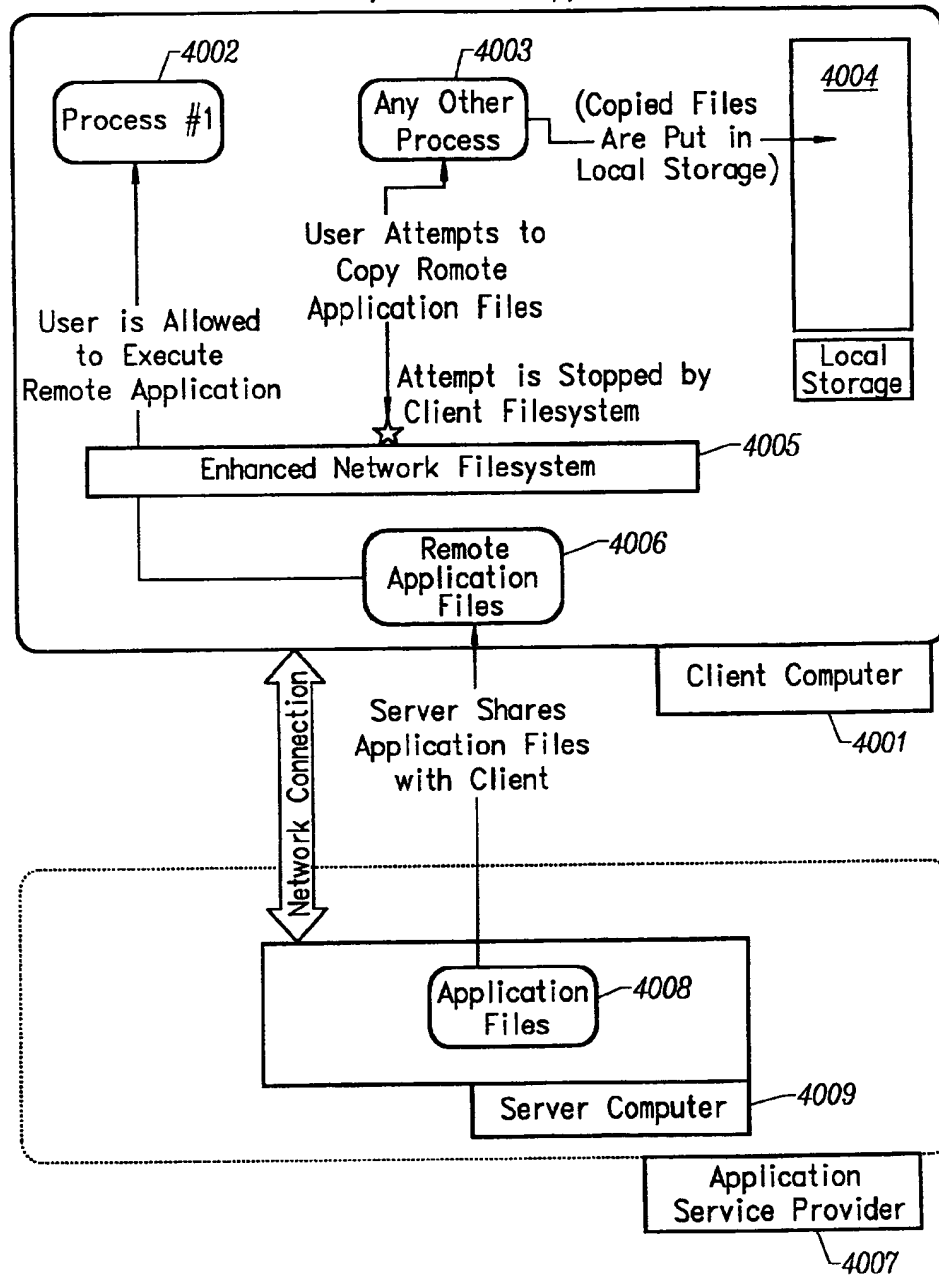


FIG. 40

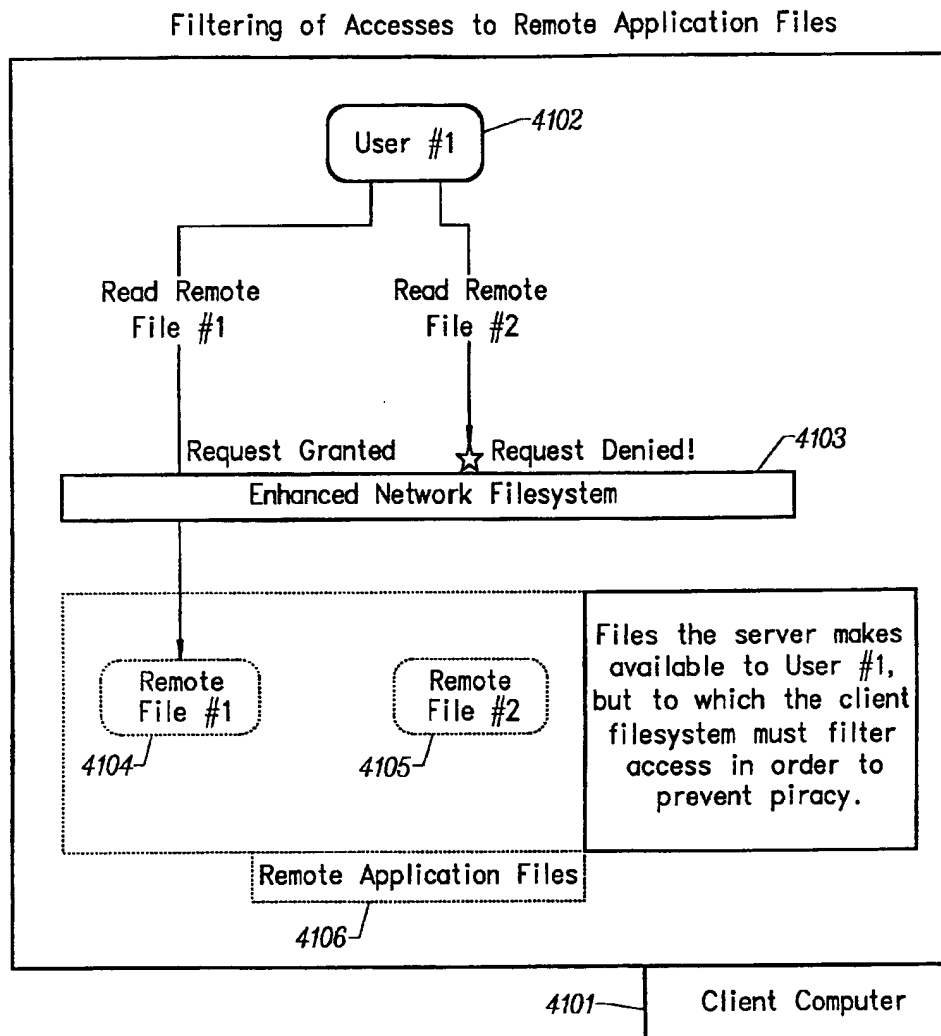


FIG. 41

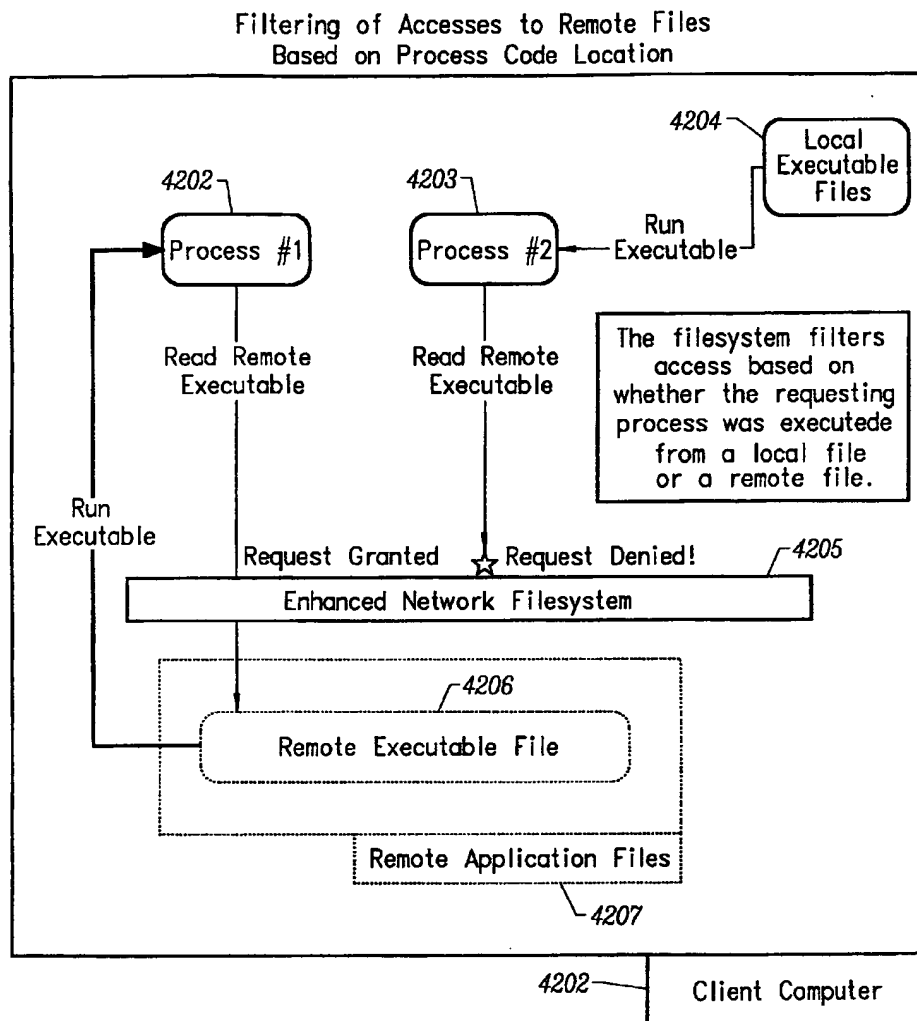


FIG. 42

Filtering of Accesses to Remote Files
Based on Targeted File Section

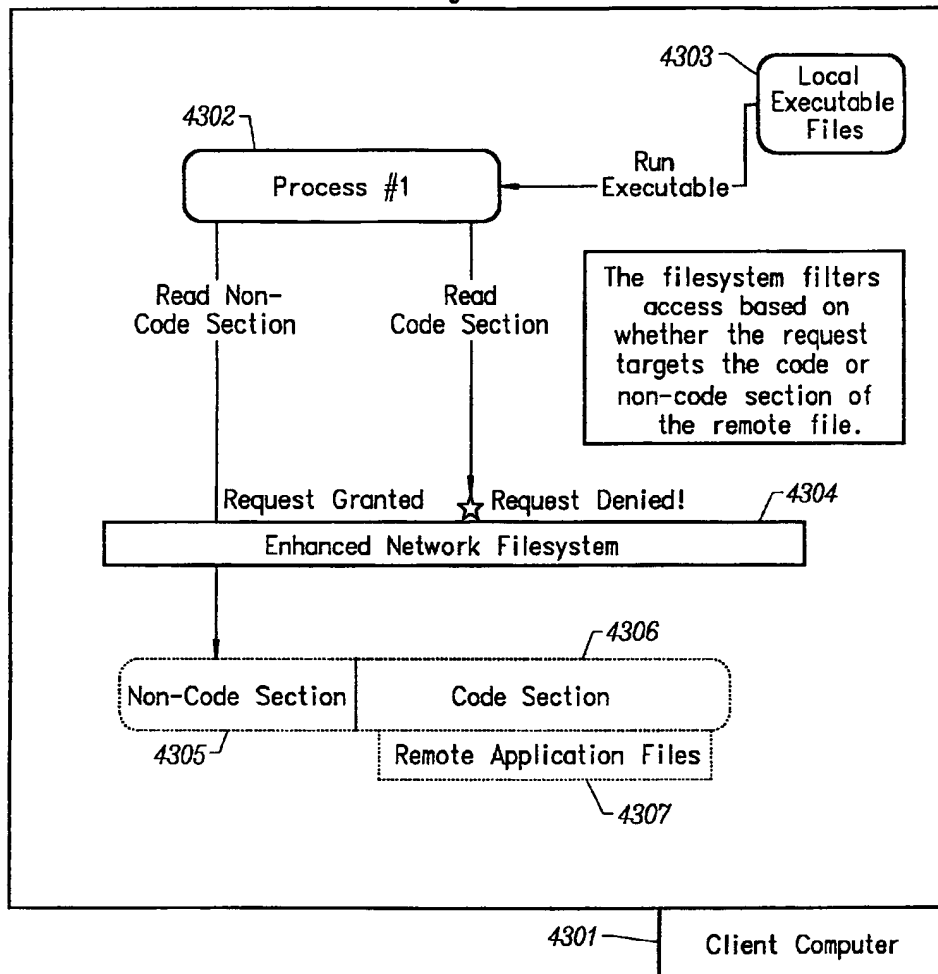


FIG. 43

Filtering of Accesses to Remote Files
Based on Surmised Purpose

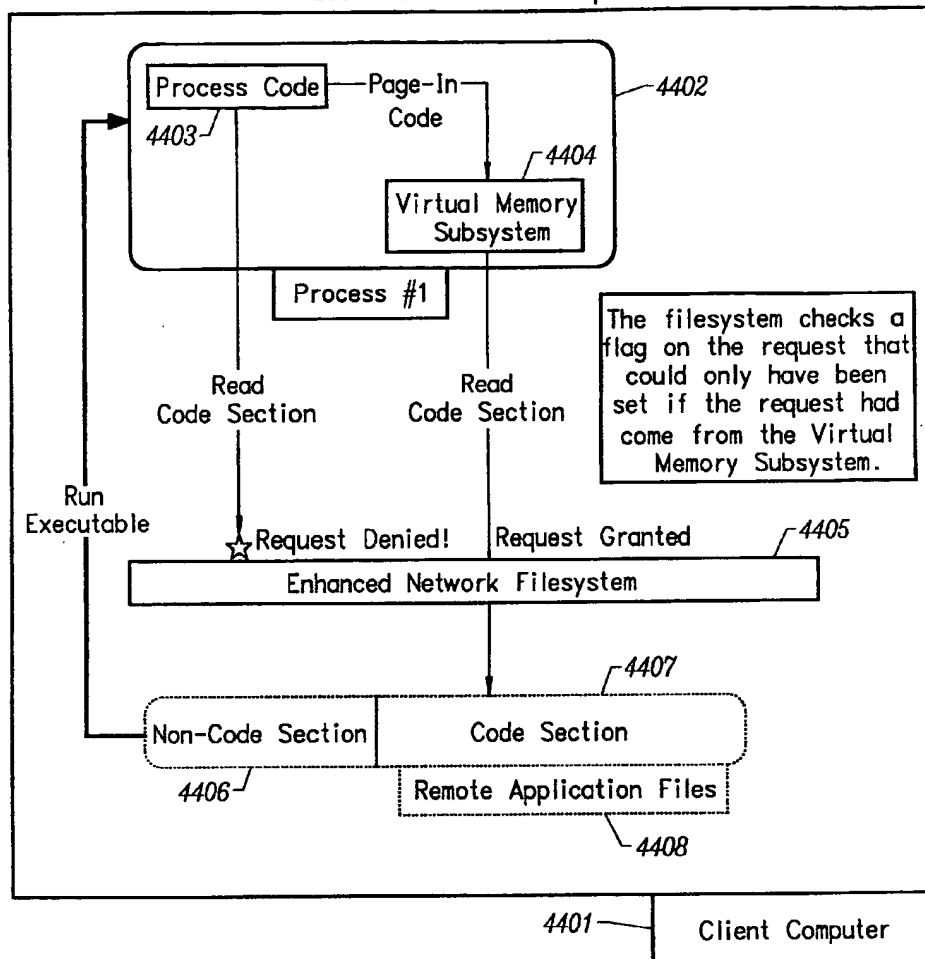


FIG. 44

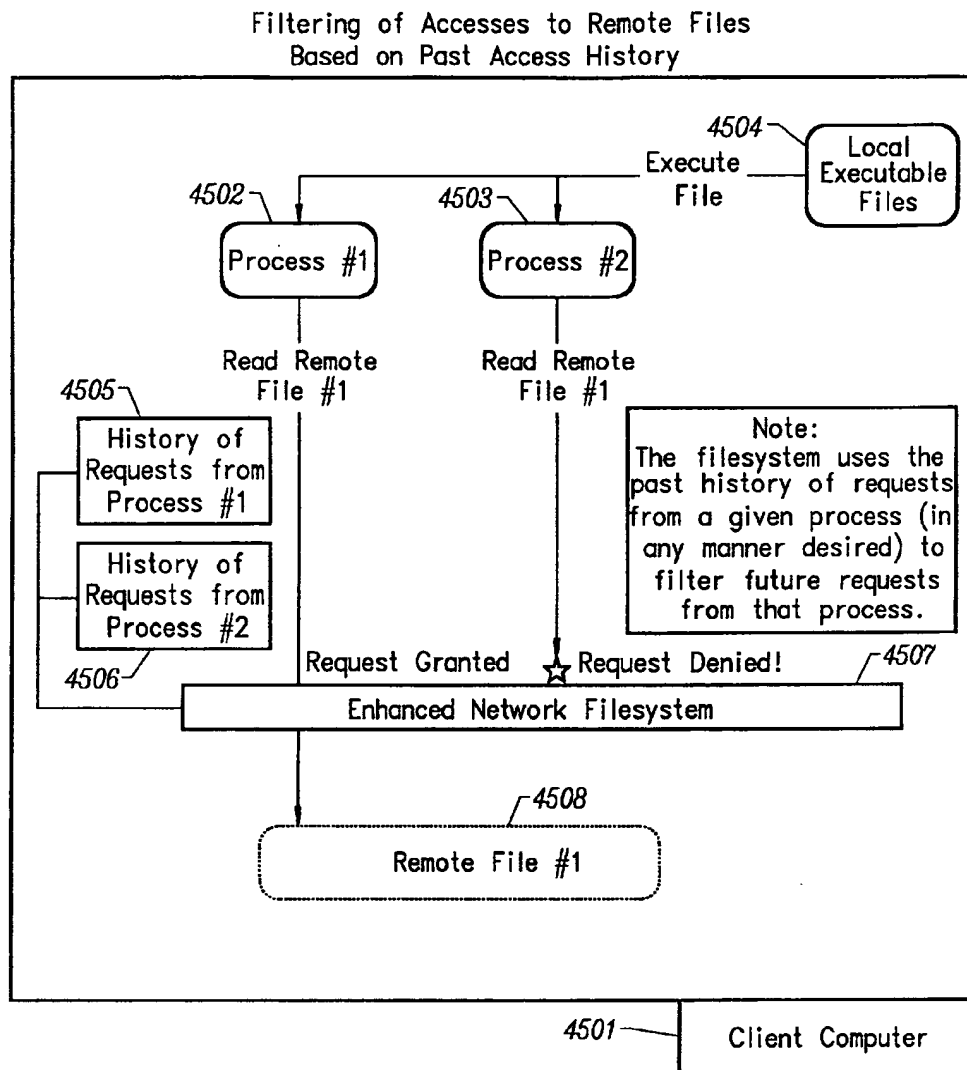


FIG. 45

CONVENTIONALLY CODED APPLICATION CONVERSION SYSTEM FOR STREAMED DELIVERY AND EXECUTION

CROSS REFERENCE TO RELATED APPLICATIONS

[0001] This application Claims benefit of U.S. Provisional Patent Application Serial No. 60/246,384, filed on Nov. 6, 2000 (OTI.2000.0).

BACKGROUND OF THE INVENTION

[0002] 1. Technical Field

[0003] The invention relates to the streaming of computer program object code across a network in a computer environment. More particularly, the invention relates to streaming and execution of existing applications across a network of servers streaming computer program object code and other related data to clients in a computer environment.

[0004] 2. Description of the Prior Art

[0005] Retail sales models of computer application programs are fairly straight forward. The consumer either purchases the application program from a retailer that is either a brick and mortar or an ecommerce entity. The product is delivered to the consumer in a shrink-wrap form.

[0006] The consumer installs the program from a floppy disk or a CD-ROM included in the packaging. A serial number is generally provided that must be entered at installation or the first time the program is run. Other approaches require that the CD-ROM be present whenever the program is run. However, CD-ROMs are easily copied using common CDR technology.

[0007] Another approach is for the consumer to effectuate the purchase through an ecommerce entity. The application program is downloaded in its entirety to the consumer across the Internet. The consumer is emailed a serial number that is required to run the program. The consumer enters the serial number at the time the program is installed or the first time the program is run.

[0008] Once the application program is installed on a machine, it resides on the machine, occupying precious hard disk space, until it is physically removed. The installer portion of the program can also be installed on a server along with the installation files. Users within an intranet can install the program from the server, across the network, onto their machines. The program is a full installation of the program and resides on the user's machine until it is manually removed.

[0009] Trial versions of programs are also available online that are a partial or full installation of the application program. The program executes normally for a preset time period. At the end of the time period, the consumer is told that he must purchase the program and execution is terminated. The drawback to this approach is that there is an easy way for the consumer to fool the program. The consumer simply uninstalls the program and then reinstalls it, thereby restarting the time period.

[0010] Additionally, piracy problems arise once the application program is resident on the consumer's computer. Serial numbers for programs are easily obtained across the Internet. Software companies lose billions of dollars a year in revenue because of this type of piracy.

[0011] The above approaches fail to adequately protect software companies' revenue stream. These approaches also require the consumer to install a program that resides indefinitely on the consumer's hard disk, occupying valuable space even though the consumer may use the program infrequently.

[0012] The enterprise arena allows Application Service Providers (ASP) to provide browser-based implementations such as Tarantella offered by Santa Cruz Operation, Inc. in Santa Cruz, Calif. and Metaframe offered by Citrix Systems Inc. of Fort Lauderdale, Fla. A remote application portal site allows the user to click on an application in his browser to execute the application. The application runs on the portal site and GUI interfaces such as display, keystrokes and mouse clicks are transferred over the wire. The access to the program is password protected. This approach allows the provider to create an audit trail and to track the use of an application program. AppStream Inc. of Palo Alto, Calif. uses Java code streamlets to provide streaming applications to the user. The system partitions a Web application program into Java streamlets. Java streamlets are then streamed to the user's computer on an as-needed basis. The application runs on the user's computer, but is accessed through the user's network browser.

[0013] The drawback to the browser-based approaches is that the user is forced to work within his network browser, thereby adding another layer of complexity. The browser or Java program manages the application program's run-time environment. The user loses the experience that the software manufacturer had originally intended for its product including features such as application invocation based on file extension associations.

[0014] It would be advantageous to provide a conventionally coded application conversion system for streamed delivery and execution that converts a conventionally coded application program into a streamed application suitable for concurrent execution on a client while being streamed from a server. It would further be advantageous to provide a conventionally coded application conversion system for streamed delivery and execution that does not require the conventionally coded application program to be recompiled or recoded.

SUMMARY OF THE INVENTION

[0015] The invention provides a conventionally coded application conversion system for streamed delivery and execution. The system converts a conventionally coded application program into a streamed application suitable for concurrent execution on a client while being streamed from a server. In addition, the invention provides a system that does not require the conventionally coded application program to be recompiled or recoded.

[0016] The invention converts locally installable applications into a data set suitable for streaming over a network. The invention monitors two classes of information during an application installation on a local computer system: system registry modifications and file modifications.

[0017] To monitor system registry modifications, the invention records the modification data when the installation program writes to the registry of the local computer system, including registry key path, value name and value data is

recorded. File modification data are logged each time an installation program modifies a file on the system. The invention sorts this data and removes duplicates and parameterizes all local-system-specific registry keys, value names, and values, so they can be recognized by the client installation software.

[0018] This data is used to create an initialization data set which is the first set of data to be streamed from the server to the client. This data set contains the information captured needed by the client to prepare the client machine for streaming a particular application. A runtime data set is also created that contains the rest of the data that is streamed to the client once the client machine is initialized for a particular application. The runtime data consists of all the regular application files and the directories containing information about those application files. Each directory contains list of file name, file number, and the metadata associated with the files in that particular directory.

[0019] A versioning table contains a list of root file numbers and version numbers. This information is used to track application patches and upgrades. Each entry in the versioning table corresponds to one patch level of the application with a corresponding new root directory. The client is sent the versioning table and compares it with the client's application root file number to find the files required for a software patch or update.

[0020] The invention monitors a running application that is being configured for a particular working environment on the local computer system. Common configuration modifications are captured and the data acquired are used to duplicate the same configuration on multiple client machines, making it unnecessary for each user to configure his/her own application installation.

[0021] The invention monitors and profiles the sequence of file blocks accessed by the application program as it runs. This information is used so that frequently used file blocks can be streamed to the client machine before other less used file blocks and cached locally on the client cache before the user starts using the streamed application for the first time. Additionally, frequently accessed files can be reordered in the directories to allow faster lookup of the file information. Finally, the association of a set of file blocks with a particular user input allows the client to prefetch from the server, a set of file blocks needed to respond to that particular user command.

[0022] Other aspects and advantages of the invention will become apparent from the following detailed description in combination with the accompanying drawings, illustrating, by way of example, the principles of the invention.

BRIEF DESCRIPTION OF THE DRAWINGS

[0023] FIG. 1 is a block schematic diagram of a preferred embodiment of the invention showing components on the server that deal with users subscribing to and running applications according to the invention;

[0024] FIG. 2 is a block schematic diagram of a preferred embodiment of the invention showing the client components supporting application delivery and execution according to the invention;

[0025] FIG. 3 is a block schematic diagram of a preferred embodiment of the invention showing the components needed to install applications on the client according to the invention;

[0026] FIG. 4 is a block schematic diagram of the Builder that takes an existing application and extracts the Application File Pages for that application according to the invention;

[0027] FIG. 5a is a block schematic diagram illustrating how the Client Network Spoofer is used to handle mapping TCP interfaces to HTTP interfaces according to the invention;

[0028] FIG. 5b is a block schematic diagram illustrating how the Client Network Spoofer is used to handle mapping TCP interfaces to HTTP interfaces according to the invention;

[0029] FIG. 6a is a block schematic diagram showing several different components of the client software according to the invention;

[0030] FIG. 6b is a block schematic diagram showing the use of volatile and non-volatile storage of code and data in the client and server according to the invention;

[0031] FIG. 7a is a block schematic diagram showing one of two ways in which data may be compressed while in transit between the server and client according to the invention;

[0032] FIG. 7b is a block schematic diagram showing the other way in which data may be compressed while in transit between the server and client according to the invention;

[0033] FIG. 8 is a block schematic diagram showing an organization of the streaming client software according to the invention;

[0034] FIG. 9 is a block schematic diagram showing an alternative organization of the streaming client software according to the invention;

[0035] FIG. 10 is a block schematic diagram showing the application streaming software consisting of a streaming block driver according to the invention;

[0036] FIG. 11 is a block schematic diagram showing the application streaming software has been divided into a disk driver and a user mode client according to the invention;

[0037] FIG. 12 is a block schematic diagram showing the unencrypted and encrypted client caches according to the invention;

[0038] FIG. 13 is a block schematic diagram showing an application generating a sequence of code or data requests to the operating system according to the invention;

[0039] FIG. 14 is a block schematic diagram showing server-based prefetching according to the invention;

[0040] FIG. 15 is a block schematic diagram showing a client-to-client communication mechanism that allows local application customization to travel from one client machine to another without involving server communication according to the invention;

[0041] FIG. 16 is a block schematic diagram showing a client cache with extensions for supporting local file customization according to the invention;

[0042] FIG. 17 is a block schematic diagram showing aspects of a preferred embodiment of the invention related to load balancing and hardware fail over according to the invention;

[0043] FIG. 18 is a block schematic diagram showing the benefits to the use of compression in the streaming of Application File Pages according to the invention;

[0044] FIG. 19 is a block schematic diagram showing pre-compression of Application File Pages according to the invention;

[0045] FIG. 20 is a block schematic diagram showing multi-page compression of Application File Pages according to the invention;

[0046] FIG. 21 is a block schematic diagram showing profile-based prefetching according to the invention;

[0047] FIG. 22 is a block schematic diagram showing the use of tokens and a License Server according to the invention;

[0048] FIG. 23 is a block schematic diagram showing a flowchart for the Builder Install Monitor according to the invention;

[0049] FIG. 24 is a block schematic diagram showing a flowchart for the Builder Application Profiler according to the invention;

[0050] FIG. 25 is a block schematic diagram showing a flowchart for the Builder SAS Packager according to the invention;

[0051] FIG. 26a is a block schematic diagram showing versioning support according to the invention;

[0052] FIG. 26b is a block schematic diagram showing versioning support according to the invention;

[0053] FIG. 27 is a block schematic diagram showing a data flow diagram for the Streamed Application Set Builder according to the invention;

[0054] FIG. 28 is a block schematic diagram showing the Streamed Application Set format according to the invention;

[0055] FIG. 29 is a block schematic diagram showing an SAS client using a device driver paradigm according to the invention;

[0056] FIG. 30 is a block schematic diagram showing an SAS client using a file system paradigm according to the invention;

[0057] FIG. 31a through 31h is a schematic diagram showing various components of the AppinstallBlock format according to the invention;

[0058] FIG. 32 is a block schematic diagram showing the Application Install Block lifecycle according to the invention;

[0059] FIG. 33 is a block schematic diagram showing peer caching according to the invention;

[0060] FIG. 34 is a block schematic diagram showing proxy caching according to the invention;

[0061] FIG. 35 is a block schematic diagram showing multicast within a LAN and a packet protocol according to the invention;

[0062] FIG. 36 is a block schematic diagram showing concurrent requests for application server pages, for the case when the page is first requested through the proxy according to the invention;

[0063] FIG. 37 is a block schematic diagram showing concurrent requests for application server pages, for the case when the page is first requested through the peer caching according to the invention;

[0064] FIG. 38 is a block schematic diagram showing concurrent requests for application server pages, for the case when the page is received only through peer caching according to the invention;

[0065] FIG. 39 is a block schematic diagram showing a client-server system using peer and proxy caching according to the invention;

[0066] FIG. 40 is a block schematic diagram showing a preferred embodiment of the invention preventing piracy of remotely served, locally executed applications according to the invention;

[0067] FIG. 41 is a block schematic diagram showing the filtering of accesses to remote application files according to the invention;

[0068] FIG. 42 is a block schematic diagram showing the filtering of accesses to remote files based on process code location according to the invention;

[0069] FIG. 43 is a block schematic diagram showing the filtering of accesses to remote files based on targeted file section according to the invention;

[0070] FIG. 44 is a block schematic diagram showing the filtering of accesses to remote files based on surmised purpose according to the invention; and

[0071] FIG. 45 is a block schematic diagram showing the filtering of accesses to remote files based on past access history according to the invention.

DETAILED DESCRIPTION OF THE INVENTION

[0072] The invention is embodied in a conventionally coded application conversion system for streamed delivery and execution in a computer environment. A system according to the invention converts a conventionally coded application program into a streamed application suitable for concurrent execution on a client while being streamed from a server. In addition, the invention provides a system that does not require the conventionally coded application program to be recompiled or recoded.

[0073] The invention provides a highly efficient and secure application delivery system in conjunction with the adaptively optimized execution of applications across a network such as the Internet, a corporate intranet, or a wide area network. This is done in such a way that existing applications do not need to be recompiled or recoded. Furthermore, the invention is a highly scalable, load-balancing, and fault-tolerant system.

[0074] When using the invention, an end-user requests applications that are resident on remote systems to be launched and run on the end-user's local system. The end-user's local system is called the client or client system, e.g., a desktop, laptop, palmtop, or information appliance. A remote system is called a server or server system and is located within a collection of one or more servers called a server cluster.

[0075] From the point of view of the client system, the application appears to be installed locally on the client even though it was initially installed on a different computer system. The applications execute locally on the client system and not on the server system. To achieve this result, the application is converted into a form suitable for streaming over the network. The streaming-enabled form of an application is called the Streamed Application Set (SAS) and the conversion process is termed the SAS Builder. The conversion of an application into its SAS form typically takes place on a system different from either an end-user client system or an Application Service Provider Server Cluster. This system is called the SAS Conversion System or, simply, the conversion system.

[0076] Components of the invention are installed on the client system to support activities such as the installation, invocation, and execution of a SAS-based application. Other components of the invention are installed on the server system to support activities such as the verification of end user application subscription and license data and the transfer and execution of a SAS-based application on the client system. Some of the client and some of the server components run in the kernel-mode while other components run in the usual user-mode.

[0077] The term Application Service Provider (ASP) refers to an entity that uses the server components on one or more server systems, i.e., an ASP Server Cluster, to deliver applications to end-user client systems. Such an entity could be, for example, a software manufacturer, an e-commerce vendor that rents or leases software, or a service department within a company. The invention enables an ASP to deliver applications across a network, in a highly efficient and secure way; the applications are adaptively optimized for execution on an end-user's client system.

[0078] A number of techniques are employed to increase the overall performance of the delivery of an application and its subsequent execution by minimizing the effect of network latency and bandwidth. Among the techniques employed are: the SAS Builder identifies sequences of frequently accessed application pages and uses this information when generating a SAS; individual SAS pages and sequences of SAS pages are compressed and cached in an in-memory cache on the server system; various aspects of the applications are monitored during their actual use on a client and the resulting profiling data is used by the client to pre-fetch (pull) and by the server to send (push) additional pages which have a high likelihood of being used prior to their actual use; and SAS pages are cached locally on a client for their immediate use when an application is invoked.

[0079] Aggregate profile data for an application, obtained by combining the profile data from all the end-user client systems running the application, is used to increase the system performance as well. A number of additional caching techniques that improve both system scalability and performance are also employed. The above techniques are collectively referred to as collaborative caching.

[0080] In an embodiment of the invention, the SAS Builder consists of three phases: installation monitoring, execution profiling, and application stream packaging. In the final SAS Builder phase, the Application Stream Packager takes the information gathered by the Application Install Monitor and the Application Execution Profiler and creates the SAS form of the application, which consists of a Stream Enabled Application Pages File and a Stream Enabled Application Install Block.

[0081] The Stream Enabled Application Install Block is used to install a SAS-based application on a client system while selected portions of the Stream Enabled Application Pages File are streamed to a client to be run on the client system. The Stream Enabled Application Install Block is the first set of data to be streamed from the server to the client and contains, among other things, the information needed by the client system to prepare for the streaming and execution of the particular application. Individual and aggregate client dynamic profile data is merged into the existing Stream Enabled Application Install Block on the server to optimize subsequent streaming of the application.

[0082] The invention employs a Client Streaming File System that is used to manage specific application-related file accesses during the execution of an application. For example, there are certain shared library files, e.g., "foo.dll", that need to be installed on the local file system, e.g., "c:\winnt\system32\foo.dll", for the application to execute. Such file names get added to a "spool database". For the previous example, the spool database would contain an entry saying that "c:\winnt\system32\foo.dll" is mapped to "z:\word\winnt\system32\foo.dll" where "z:" implies that this file is accessed by the Client Streaming File System. The Client Spoofer will then redirect all accesses to "c:\winnt\system32\foo.dll" to "z:\word\winnt\system32\foo.dll". In this manner, the client system gets the effect of the file being on the local machine whereas in reality the file is streamed from the server. Several different classes of files can be treated in this way, e.g., specific application registry entries and application-based networking calls when such calls cross a firewall.

[0083] Lastly, the invention incorporates a number of software anti-piracy techniques directed at combating the piracy of applications of the type described herein that are delivered to the end-user over a network for execution on a client system. Among the anti-piracy techniques included are: client-side fine-grained filtering of file accesses directed at remotely served files; filtering of file accesses based on where the code for the process that originated the request is stored; identification of crucial portions of application files and filtering file access depending on the portions of the application targeted; filtering of file accesses based on the surmised purpose of the file access, as determined by examining the program stack or flags associated with the request; and filtering of file accesses based on the surmised purpose of the file access, as determined by examining a history of previous file accesses by the same process.

[0084] As mentioned above, the invention provides server and client technology for streaming application delivery and execution. The invention includes secure license-based streaming delivery of applications over Internet/extranets/intranets utilizing client-based execution with client caching and server-based file accesses by page.

[0085] 1. The invention provides many advantages over the present approaches, including:

[0086] Secure license-based streaming delivery over Internet/extranets/intranets:

[0087] reduces IT costs over client installation;

[0088] supports rental model of app delivery, which opens new markets and increases user convenience over purchase and client installation; and

[0089] enhances the opportunities to prevent software piracy over purchase and client installation.

[0090] Client-based execution with client caching:

- [0091]** increases typical application performance over server-based execution;
- [0092]** reduces network latency and bandwidth usage over non-cached client execution; and
- [0093]** allows existing applications to be run w/o rewrite/recompile/rebuild unlike other explicitly-distributed client/server application delivery approaches.

[0094] Server-based file accesses:

- [0095]** improve server-scaling over server-based execution;
- [0096]** allow transparent failover to another server whereas server-based execution does not;
- [0097]** make server load balancing easier than it is with server-based execution; and
- [0098]** allow increased flexibility in server platform selection over server-based execution.

[0099] Server-based file accesses by page:

- [0100]** reduce network latency over complete file downloads;
- [0101]** reduce network bandwidth overhead over complete file downloads; and
- [0102]** reduce client cache footprint over complete file downloads.

[0103] 2. Features of the Invention**[0104]** A) Server Components Supporting Application Delivery and Execution**[0105]** i) referring to FIG. 1, the server components include:

- [0106]** a. Client/server network interface **110** that is common to the client **113** and the server. This is the communication mechanism through which the client and the server communicate.
- [0107]** b. The Subscription Server **105**—This is the server the client **113** connects to for subscribing and unsubscribing applications. This server then adds/deletes the subscription information to the Subscription Database **101** and also updates the License Database **102** with the information stating that the client **113** can/cannot run the subscribed information under the agreed upon licensing terms. This communication between the client **113** and the Subscription Server **105** happens over SSL that is an industry standard protocol for secure communication. The Subscription Server **105** is also contacted for updating any existing subscription information that is in the Subscription Database **101**.
- [0108]** c. The License Server **106**—This is the server the client **113** connects to for getting a license to run an application after the client has subscribed to the application. This server validates the user and his subscriptions by consulting the License Database **102**. If the client **113** does have a valid license, the License Server **106** sends an "Access token" to the

client that is encrypted using an "encryption key" that the License Database **102** obtains from the Server Config Database **103**. The "Access token" contains information like the Application ID and an expiration time. Along with the "Access token," the License Server **106** also sends a list of least loaded application servers that it obtains from the Server Config Database **103** and also the expiration time that was encoded in the "Access token". The client **113** uses this expiration time to know when to ask for a new token. This communication between the client **113** and the License Server **106** happens over SSL.

[0109] d. The Application Server **107**—Once the client **113** obtains an "Access token" to run an application, it connects to the Application Server **107** and presents to it the "Access token" along with the request for the application bits. Note that the "Access token" is opaque to the client **113** since it does not have the key to decrypt it. The Application Server **107** validates the "Access token" by decrypting it using a "decryption key" obtained from the Server Config Database **103** and checking the content against a predefined value like for example the Application ID and also by making sure that the expiration time in the "Access token" has not elapsed. It then serves the appropriate bits to the client **113** to enable it to run the application. The encryption and decryption keys could be something like a private key/public key pair or a symmetric key or any other means of providing security. Note that the keys are uniform across all the servers within an ASP.

[0110] e. The Monitor Server **108**—It monitors the load in terms of percent of CPU utilization on the Application Servers **107** and the License Servers **106** on a periodic basis (for example—every minute) and adds that information to the Server Config Database **103**.

[0111] f. The Profile Server **109**—it receives profile information from the clients periodically. It adds this information to the Profile Database **104**. The Profile Server **109** based on the profile information from different clients updates the App Prefetch Page List section of the Stream App Install Blocks **112**.

[0112] ii) The data structures supporting the above server components include:

[0113] a. Subscription Database **101**—This is the database that stores the user information in terms of username, list of apps subscribed, password, billing information, address, group, admin. The username is the primary key. The admin field identifies if this user has admin privileges for the group he belongs to.

[0114] b. License Database **102**—This is the database that stores licensing information, i.e., which user can run what application and under which license. This database also keeps track of usage information, i.e., which user has used which application for how long and how many times. The information looks like:

[0115] Username, application, time of usage, number of times run

- [0116] Username, application, licensing policy
- [0117] Username, application, is app running, no of instances, time of start The username is the primary key. The licensing policy could be something simple like expiry date or something more complicated like number of instances simultaneously allowed within a group etc.
- [0118] c. Server Config Database 103—This database stores information about which server can run which application, what is the load on all the servers, what is the encryption “key” to be used by the servers and all other information that is needed by the servers. The information looks like:
- [0119] Server IP address, App/Slim server, application list, current load
- [0120] Encryption key, Decryption key The Server IP address is the primary key for the first table. The keys are common across all servers.
- [0121] d. Profile Database 104—This database stores the profile information received by the profile server from the clients periodically. The information looks like:
- [0122] Application ID, File ID, Block ID number of hits The Application ID is the primary key.
- [0123] e. Application File Pages 111—This is the one of the outputs of the “builder” as explained below and is put on the Application Server 107 so that it can serve the appropriate bits to the client.
- [0124] f. Stream App Install Blocks 112—This is the other output of the “builder” and contains the information for successfully installing applications on the client for streaming applications.
- [0125] B) Client Components Supporting Application Delivery & Execution
- [0126] i) With respect to FIGS. 1 and 2, these client components include:
- [0127] a. Client/Server Network interface 202—This is the same interface as explained above.
- [0128] b. Client License Manager 205—This component requests licenses (“Access tokens”) from the License Server 106 when the client wants to run applications. The License Server 106 sends an “Access token” to the client that can be used to run the applications by presenting it to the Application Server 107. Along with the token, the License Server 106 also sends the expiry time of the token. The Client License Manager 205 renews the token just before the expiry period so that the client can continue running the application. When the application is complete, the Client License Manager 205 releases the token by sending a message to the License Server 106. In addition, when a user has subscribed to an application, the Client License Manager 205 first checks to make sure that the application is installed on the machine the user is trying to run the application from and if not requests for the application installation. It does this using a list of Installed Apps that it maintains.
- [0129] c. Client Cache Manager 207—This component caches the application bits received from the Application Server 107 so that next time a request is made to the same bits, the request can be served by the cache instead of having to go to the Application Server 107. The Client Cache Manager 207 has a limited amount of space on the disk of the client machine that it uses for the cache. When the space is fully occupied, the Client Cache Manager 207 uses a policy to replace existing portions of the cache. This policy can be something like LRU, FIFO, random etc. The Client Cache Manager 207 is responsible for getting the application bits requested by the Client Streaming File System 212. If it does not have the bits cached, it gets them from the Application Server 107 through the network interface. However it also need to get the “Access token” from the Client License Manager 205 that it needs to send along with the request for the application bits. The Client Cache Manager 207 also updates the Prefetch History Info 209 with the requests it receives from the Client Streaming File System 212.
- [0130] d. Client Streaming File System 212—This component serves all file system requests made by the application running on the client. The application makes calls like “read”, “write” etc. to files that need to be streamed. These requests lead to page faults in the operating system and the page faults are handled by the Client Streaming File System 212 that in turn asks the Client Cache Manager 207 for the appropriate bits. The Client Cache Manager 207 will send those bits from the cache if they exist there or forward the request to the Application Server 107 through the network interface to get the appropriate bits.
- [0131] e. Client Prefetcher 208—This component monitors the requests made by the client to the Application Server 107 and uses heuristics to make additional requests to the Application Server 107 so that the bits can be obtained from the Application Server 107 before the client machine makes the request for them. This is mainly to hide the latency between the client and the Application Server 107. The history information of the requests is stored in the Prefetch History Info file 209.
- [0132] f. Client Profiler 203—At specific time intervals, the client profiler sends the profile information, which is the Prefetch History Info to the prefetch server at the ASP that can then update the App Prefetch Page Lists for the different applications accordingly.
- [0133] g. Client File Spoofer 211—Certain files on the client need to be installed at specific locations on the client system. To be able to stream these files from the Application Server 107, the Client Spoofer 211 intercepts all requests to these files made by a running application and redirects them to the Client Streaming File System 212 so that the bits can be streamed from the Application Server 107.
- [0134] h. Client Registry Spoofer 213—Similar to files, certain registry entries need to be different when the application being streamed is running and

since it is undesirable to overwrite the existing registry value, the read of the registry value is redirected to the Client Registry Spoofer 215 which returns the right value. However, this is optional as it is very likely that overwriting the existing registry value will make the system work just fine.

- [0135] i. Client Network Spoofer 213—Certain applications make networking calls through a protocol like TCP. To make these applications work across firewalls, these networking calls need to be redirected to the Client Network Spoofer 213 which can tunnel these requests through a protocol like HTTP that works through firewalls.
- [0136] ii) The data structures needed to support the above client components include:
 - [0137] a. File Spoof Database 210—The list of files the requests to which need to be redirected to the Client Streaming File System 212. This information looks like (The source file name is the primary key)
 - [0138] Source File Name, Target File Name
 - [0139] b. Registry Spoof Database 216—List of registry entries and their corresponding values that need to be spoofed. Each entry looks like:
 - [0140] Registry entry, new value
 - [0141] c. Network Spoof Database 214—Like of IP addresses, the networking connections to which need to be redirected to the Client Network Spoofer 213. Each entry looks like (IP address is the primary key):
 - [0142] IP address, Port number, new IP address, new Port number
 - [0143] d. Client Stream Cache 206—The on-disk cache that persistently stores application bits.
 - [0144] e. Known ASPs and Installed Apps 204—The list of ASP servers (Application, License and Subscription) and also the list of applications that are installed on the client.
 - [0145] f. Prefetch History Info 209—The history of the requests made to the cache. This consists of which blocks were requested from which file for which application and how many times each block was requested. It also consists of predecessor-successor information indicating which block got requested after a particular block was requested.
- [0146] C) Client Application Installation
- [0147] Referring to FIG. 3, the client application installation components include:
 - [0148] i) Client License Manager 303—This is the same component explained above.
 - [0149] ii) Client Application Installer 305—This component is invoked when the application needs to be installed. The Client Application Installer 305 sends a specific request to the Application Server 107 for getting the Stream App Install Block 301 for the particular application that needs to be installed. The Stream App Install Block 301 consists of the App Prefetch Page List 306, Spoof Database 308, 309, 310, and App Registry Info 307. The Client Appli-

cation Installer 305 then updates the various Spoof Databases 308, 309, 310 and the Registry 307 with this information. It also asks the Client Prefetcher 208 to start fetching pages in the App Prefetch Page List 306 from the Application Server 107. These are the pages that are known to be needed by a majority of the users when they run this application.

[0150] D) Application Stream Builder Input/Output

[0151] With respect to FIG. 4, the Builder components include the following:

[0152] i) Application Install Monitor 403—This component monitors the installation of an application 401 and figures out all the files that have been created during installation 402, registry entries that were created and all the other changes made to the system during installation.

[0153] ii) Application Profiler 407—After the application is installed, it is executed using a sample script. The Application Profiler 407 monitors the application execution 408 and figures out the application pages that got referenced during the execution.

[0154] iii) App Stream Packager 404—The App Stream Packager 404 takes the information gathered by the Application Profiler 407 and the Application Install Monitor 403 and forms the Application File Pages 406 and the Stream App Install Block 405 from that information.

[0155] E) Network Spoofing for Client-server Applications

[0156] Referring to FIGS. 1, 4, 5a, 5b, and 6a, the component that does the Network Spoofing is the TCP to HTTP converter 503, 507. The basic idea is to take TCP packets and tunnel them through HTTP on one side and do exactly the opposite on the other. As far as the client 501 and the server 502 are concerned the communication is TCP and so existing applications that run with that assumption work unmodified. This is explained in more detail below.

[0157] On the client side, the user launches an application that resides on the Client Streaming File System. That application may be started in the same ways that applications on other client file systems may be started, e.g., opening a data file associated with the application or selecting the application from the Start/Programs menu in a Windows system. From the point of view of the client's operating system and from the point of view of the application itself, that application is located locally on the client.

[0158] Whenever a page fault occurs on behalf of any application file residing on the Client Streaming File System 604, that file system requests the page from the Client Cache Manager 606. The Client Cache Manager 606, after ensuring via interaction with the Client License Manager 608 that the user's client system holds a license to run the application at the current time, checks the Client Stream Cache 611 and satisfies the page fault from that cache, if possible. If the page is not currently in the Client Stream Cache 611, the Client Cache Manager 606 makes a request to the Client/Server Network Interface 505, 609 to obtain that page from the Application File Pages stored on an Application Server 506.

[0159] The Client Prefetcher 606 tracks all page requests passed to the Client Cache Manager 606. Based on the pattern of those requests and on program locality or program history, the Client Prefetcher 606 asks the Client Cache Manager 606 to send additional requests to the Client/Server Network Interface 505, 609 to obtain other pages from the Application File Pages stored on the Application Server 506.

[0160] Files located on the Client Streaming File System 604 are typically identified by a particular prefix (like drive letter or pathname). However, some files whose names would normally imply that they reside locally are mapped to the Client Streaming File System 604, in order to lower the invention's impact on the user's local configuration. For instance, there are certain shared library files (dll's) that need to be installed on the local file system (c:\winnt\system32\foo.dll). It is undesirable to add that file on the user's system. The file name gets added to a "spoof database" which contains an entry saying that c:\winnt\system32\foo.dll is mapped to z:\word\winnt\system32\foo.dll where z: implies that it is the Client Streaming File System. The Client Spoofer 603 will then redirect all accesses to c:\winnt\system32\foo.dll to z:\word\winnt\system32\foo.dll. In this manner the client system gets the effect of the file being on the local machine whereas in reality the file is streamed from the server.

[0161] In a similar fashion the Client Spoofer 603 may also be used to handle mapping TCP interfaces to HTTP interfaces. There are certain client-server applications (like ERP/CRM applications) that have a component running on a client and another component running on a database server, Web server etc. These components talk to each other through TCP connections. The client application will make TCP connections to the appropriate server (for this example, a database server) when the client piece of this application is being streamed on a user's machine.

[0162] The database server could be resident behind a firewall and the only way for the client and the server to communicate is through a protocol like HTTP that can pass through firewalls. To enable the client to communicate with the database server, the client's TCP requests need to be converted to HTTP and sent to the database server. Those requests can be converted back to TCP so that the database server can appropriately process the requests just before the requests reach the database server. The Client Spoofer's 603 responsibility in this case is to trap all TCP requests going to the database server and convert it into HTTP requests and take all HTTP requests coming from the database server and convert them into TCP packets. Note that the TCP to HTTP converters 505, 507 convert TCP traffic to HTTP and vice versa by embedding TCP packets within the HTTP protocol and by extracting the TCP packets from the HTTP traffic. This is called tunneling.

[0163] When the Client License Manager 608 is asked about a client's status with respect to holding a license for a particular application and the license is not already being held, the Client License Manager 608 contacts the License Server 106 via the Client/Server Network Interface 609 and asks that the client machine be given the license. The License Server 106 checks the Subscription 101 and License 102 Databases and, if the user has the right to hold the license at the current time, it sends back an Access Token, which represents the right to use the license. This Access Token is renewed by the client on a periodic basis.

[0164] The user sets up and updates his information in the Subscription 101 and License 102 Databases via interacting with the Subscription Server 105.

[0165] Whenever a user changes his subscription information, the Subscription Server 105 signals the user's client system since the client's Known ASPs and Installed Apps information potentially needs updating. The client system also checks the Subscription 101 and License 102 Databases whenever the user logs into any of his client systems set up for Streaming Application Delivery and Execution. If the user's subscription list in the Subscription 101 and License 102 Databases list applications that have not been installed on the user's client system, the user is given the opportunity to choose to install those applications.

[0166] Whenever the user chooses to install an application, the Client License Manager 608 passes the request to the Client Application Installer 607 along with the name of the Stream App Install Block to be obtained from the Application Server 107. The Client Application Installer 607 opens and reads that file (which engages the Client Streaming File System) and updates the Client system appropriately, including setting up the spoof database, downloading certain needed non-application-specific files, modifying the registry file, and optionally providing a list of applications pages to be prefetched to warm up the Client Stream Cache 611 with respect to the application.

[0167] The Application Stream Builder creates the Stream App Install Block 405 used to set up a client system for Streaming Application Delivery and Execution and it also creates the set of Application File Pages 406 sent to satisfy client requests by the Application Server 107. The process that creates this information is offline and involves three components. The Application Install Monitor 403 watches a normal installation of the application and records various information including registry entries, required system configuration, file placement, and user options. The Application Profiler 407 watches a normal execution of the application and records referenced pages, which may be requested to pre-warm the client's cache on behalf of this application. The Application Stream Packager 404 takes information from the other two Builder components, plus some information it compiles with respect to the layout of the installed application and forms the App Install Block 405 and the set of Application File Pages 406.

[0168] Server fail-over and server quality of service problems are handled by the client via observation and information provided by the server components. An ASP's Subscription Server provides a list of License Servers associated with that ASP to the client, when the user initiates/modifies his account or when the client software explicitly requests a new list. A License Server provides a list of Application Servers associated with an application to the client, whenever it sends the client an Access Token for the application.

[0169] Should the client observe apparent non-response or slow response from an Application Server, it switches to another Application Server in its list for the application in question. If none of the Application Servers in its list respond adequately, the client requests a new set for the application from a License Server. The strategy is similar in the case in which the client observes apparent non-response or slow response from a License Server; the client switches to another License Server in its list for the ASP in question.

If none of the License Servers in its list responds adequately, the client requests a new set of License Servers from the ASP.

[0170] Server load balancing is handled by the server components in cooperation with the client. A server monitor component tracks the overall health and responsiveness of all servers. When a server is composing one of the server lists mentioned in the previous paragraph, it selects a set that is alive and relatively more lightly used than others. Client cooperation is marked by the client using the server lists provided by the servers in the expected way, and not unilaterally doing something unexpected, like continuing to use a server which does not appear in the most recent list provided.

[0171] Security issues associated with the server client relationship are considered in the invention. To ensure that the communication between servers and clients is private and that the servers in question are authorized via appropriate certification, an SSL layer is used. To ensure that the clients are licensed to use a requested application, user credentials (username+password) are presented to a License Server, which validates the user and his licensing status with respect to the application in question and issues an Access Token, and that Access Token is in turn presented to an Application Server, which verifies that the Token's validity before delivering the requested page. Protecting the application in question from piracy on the client's system is discussed in another section, below.

Client-side Performance Optimization

[0172] This section focuses on client-specific portions of the invention. The invention may be applied to any operating system that provides a file system interface or block driver interface. A preferred embodiment of the invention is Windows 2000 compliant.

[0173] With respect to FIG. 6a, several different components of the client software are shown. Some components will typically run as part of the operating system kernel, and other portions will run in user mode.

[0174] The basis of the client side of the streamed application delivery and execution system is a mechanism for making applications appear as though they were installed on the client computer system without actually installing them.

[0175] Installed applications are stored in the file system of the client system as files organized in directories. In the state of the art, there are two types of file systems: local and network. Local file systems are stored entirely on media (disks) physically resident in the client machine. Network file systems are stored on a machine physically separate from the client, and all requests for data are satisfied by getting the data from the server. Network file systems are typically slower than local file systems. A traditional approach to use the better performance of a local file system is to install important applications on the local file system, thereby copying the entire application to the local disk. The disadvantages of this approach are numerous. Large applications may take a significant amount of time to download, especially across slower wide area networks. Upgrading applications is also more difficult, since each client machine must individually be upgraded.

[0176] The invention eliminates these two problems by providing a new type of file system: a streaming file system. The streaming file system allows applications to be run

immediately by retrieving application file contents from the server as they are needed, not as the application is installed. This removes the download cost penalty of doing local installations of the application. The streaming file system also contains performance enhancements that make it superior to running applications directly from a network file system. The streaming file system caches file system contents on the local machine. File system accesses that hit in the cache are nearly as fast as those to a local file system. The streaming file system also has sophisticated information about application file access patterns. By using this knowledge, the streaming file system can request portions of application files from the server in advance of when they will actually be needed, thus further improving the performance of applications running on the application streaming file system.

[0177] In a preferred embodiment of the invention, the application streaming file system is implemented on the client using a file system driver and a helper application running in user mode. The file system driver receives all requests from the operating system for files belonging to the application streaming file system. The requests it handles are all of the standard file system requests that every file system must handle, including (but not limited to) opening and closing files, reading and writing files, renaming files, and deleting files. Each file has a unique identifier consisting of an application number, and a file number within that application. In one embodiment of the invention, the application number is 128 bits and the file number is 32 bits, resulting in a unique file ID that is 160 bits long. The file system driver is responsible for converting path names (such as "z:\program files\foo.exe") into file IDs (this is described below). Once the file system driver has made this translation, it basically forwards the request to the user-mode program to handle.

[0178] The user-mode program is responsible for managing the cache of application file contents on the local file system and contacting the application streaming server for file contents that it cannot satisfy out of the local cache. For each file system request, such as read or open, the user-mode process will check to see if it has the requested information in the cache. If it does, it can copy the data from the cache and return it to the file system driver. If it does not, it contacts the application streaming server over the network and obtains the information it needs. To obtain the contents of the file, the user-mode process sends the file identifier for the file it is interested in reading along with an offset at which to read and the number of bytes to read. The application streaming server will send back the requested data.

[0179] The file system can be implemented using a fragmented functionality to facilitate development and debugging. All of the functionality of the user-mode component can be put into the file system driver itself without significantly changing the scope of the invention. Such an approach is believed to be preferred for a client running Windows 95 as the operating system.

[0180] Directories are specially formatted files. The file system driver reads these from the user mode process just like any other files with reads and writes. Along with a header containing information about the directory (such as how long it is), the directory contains one entry for each file that it contains. Each entry contains the name of the file and its file identifier. The file identifier is necessary so that the specified file can be opened, read, or written. Note that since directories are files, directories may recursively contain

other directories. All files in an application streaming file system are eventual descendents of a special directory called the "root". The root directory is used as the starting point for parsing file names.

[0181] Given a name like "z:/foo/bar/baz", the file system driver must translate the path "z:/foo/bar/baz" into a file identifier that can be used to read the file from the application streaming service. First, the drive letter is stripped off, leaving "/foo/bar/baz". The root directory will be searched for the first part of the path, in this case "foo". If the file "foo" is found in the root directory, and the file "foo" is a directory, then "foo" will be searched for the next portion of the path, "bar". The file system driver achieves this by using the file id for "foo" (found by searching the root directory) to open the file and read its contents. The entries inside "foo" are then searched for "bar", and this process continues until the entire path is parsed, or an error occurs.

[0182] In the following examples and text, the root directory is local and private to the client. Each application that is installed will have its own special subdirectory in the root directory. This subdirectory will be the root of the application. Each application has its own root directory.

[0183] The invention's approach is much more efficient than other approaches like the standard NFS approach. In those cases, the client sends the entire path "/foo/bar/baz" is sent to the server and the server returns the file id for that file. Next time there is a request for "/foo/bar/baz2" again the entire path needs to be sent. In the approach described here, once the request for "bar" was made, the file ids for all files within bar are sent back including the ids for "baz" and "baz2" and hence "baz2" will already be known to client. This avoids communication between the client and the server.

[0184] In addition, this structure also allows applications to be easily updated. If certain code segments need to be updated, then the code segment listing in the application root directory is simply changed and the new code segment subdirectory added. This results in the new and correct code segment subdirectory being read when it is referenced. For example if a file by the name of "/foo/bar/baz3" needs to be added, the root directory is simply changed to point to a new version of "foo" and that new version of "foo" points to a new version of "bar" which contains "baz3" in addition to the files it already contained. However the rest of the system is unchanged.

[0185] Client Features

[0186] Referring to FIGS. 6a and 6b, a key aspect of the preferred embodiment of the invention is that application code and data are cached in the client's persistent storage 616, 620. This caching provides better performance for the client, as accessing code and data in the client's persistent storage 620 is typically much faster than accessing that data across a wide area network. This caching also reduces the load on the server, since the client need not retrieve code or data from the application server that it already has in its local persistent storage.

[0187] In order to run an application, its code and data must be present in the client system's volatile storage 619. The client software maintains a cache of application code and data that normally reside in the client system's non-volatile memory 620. When the running application requires

data that is not present in volatile storage 619, the client streaming software 604 is asked for the necessary code or data. The client software first checks its cache 611, 620 in nonvolatile storage for the requested code or data. If it is found there, the code or data are copied from the cache in nonvolatile storage 620 to volatile memory 619. If the requested code or data are not found in the nonvolatile cache 611, 620, the client streaming software 604 will acquire the code or data from the server system via the client's network interface 621, 622.

[0188] Application code and data may be compressed 623, 624 on the server to provide better client performance over slow networks. Network file systems typically do not compress the data they send, as they are optimized to operate over local area networks.

[0189] FIGS. 7a & 7b demonstrate two ways in which data may be compressed while in transit between the server and client. With either mechanism, the client may request multiple pieces of code and data from multiple files at once. FIG. 7A illustrates the server 701 compressing the concatenation of A, B, C, and D 703 and sending this to the client 702. FIG. 7B illustrates the server 706 separately compressing A, B, C, and D 708 and sending the concatenation of these compressed regions to the client 707. In either case, the client 702, 707 will decompress the blocks to retrieve the original contents A, B, C, and D 704, 709 and these contents will be stored in the cache 705, 710.

[0190] The boxes marked "Compression" represent any method of making data more compact, including software algorithms and hardware. The boxes marked "Decompression" represent any method for expanding the compacted data, including software algorithms and hardware. The decompression algorithm used must correspond to the compression algorithm used.

[0191] The mechanism for streaming of application code and data may be a file system. Many network file systems exist. Some are used to provide access to applications, but such systems typically operate well over a local area network (LAN) but perform poorly over a wide area network (WAN). While this solution involves a file system driver as part of the client streaming software, it is more of an application delivery mechanism than an actual file system.

[0192] With respect to FIG. 8, application code and data are installed onto the file system 802, 805, 806, 807 of a client machine, but they are executed from the volatile storage (main memory). This approach to streamed application delivery involves installing a special application streaming file system 803, 804. To the client machine, the streaming file system 803, 804 appears to contain the installed application 801. The application streaming file system 803 will receive all requests for code or data that are part of the application 801. This file system 803 will satisfy requests for application code or data by retrieving it from its special cache stored in a native file system or by retrieving it directly from the streaming application server 802. Code or data retrieved from the server 802 will be placed in the cache in case it is used again.

[0193] Referring to FIG. 9, an alternative organization of the streaming client software is shown. The client software is divided into the kernel-mode streaming file system driver 905 and a user-mode client 902. Requests made to the

streaming file system driver 905 are all directed to the user-mode client 902, which handles the streams from the application streaming server 903 and sends the results back to the driver 905. The advantage of this approach is that it is easier to develop and debug compared with the pure-kernel mode approach. The disadvantage is that the performance will be worse than that of a kernel-only approach.

[0194] As shown in FIGS. 10 and 11, the mechanism for streaming of application code and data may be a block driver 1004, 1106. This approach is an alternative to that represented by FIGS. 8 and 9.

[0195] With respect to FIG. 10, the application streaming software consists of a streaming block driver 1004. This block driver 1004 provides the abstraction of a physical disk to a native file system 1003 already installed on the client operating system 1002. The driver 1004 receives requests for physical block reads and writes, which it satisfies out of a cache on a standard file system 1003 that is backed by a physical disk drive 1006, 1007. Requests that cannot be satisfied by the cache go to the streaming application server 1005, as before.

[0196] Referring to FIG. 11, the application streaming software has been divided into a disk driver 1106 and a user mode client 1102. In a manner similar to that of FIG. 9, the disk driver 1106 sends all requests it gets to the user-mode client 1102, which satisfies them out of the cache 1107, 1108 or by going to the application streaming server 1103.

[0197] The persistent cache may be encrypted with a key not permanently stored on the client to prevent unauthorized use or duplication of application code or data. Traditional network file systems do not protect against the unauthorized use or duplication of file system data.

[0198] With respect to FIG. 12, unencrypted and encrypted client caches. A, B, C, and D 1201 representing blocks of application code and data in their natural form are shown. $E_k(X)$ represents the encryption of block X with key k 1202. Any encryption algorithm may be used. The key k is sent to the client upon application startup, and it is not stored in the application's persistent storage.

[0199] Client-initiated prefetching of application code and data helps to improve interactive application performance. Traditional network file systems have no prefetching or simple locality based prefetching.

[0200] Referring to FIG. 13, the application 1301 generates a sequence of code or data requests 1302 to the operating system(OS) 1303. The OS 1303 directs these 1304 to the client application streaming software 1305. The client software 1305 will fetch the code or data 1306 for any requests that do not hit in the cache from the server 1307, via the network. The client software 1305 inspects these requests and consults the contents of the cache 1309 as well as historic information about application fetching patterns 1308. It will use this information to request additional blocks of code and data that it expects will be needed soon. This mechanism is referred to as "pull prefetching."

[0201] Server-initiated prefetching of application code and data helps to improve interactive application performance. Traditional network file systems have no prefetching or simple locality based prefetching.

[0202] With respect to FIG. 14, the server-based prefetching is shown. As in FIG. 13, the client application streaming software 1405 makes requests for blocks 1407 from the application streaming server 1408. The server 1408 examines the patterns of requests made by this client and selectively returns to the client additional blocks 1406 that the client did not request but is likely to need soon. This mechanism is referred to as "push prefetching."

[0203] A client-to-client communication mechanism allows local application customization to travel from one client machine to another without involving server communication. Some operating systems have a mechanism for copying a user's configuration and setup to another machine. However, this mechanism typically doesn't work outside of a single organization's network, and usually will copy the entire environment, even if only the settings for a single application are desired.

[0204] Referring to FIG. 15, a client-to-client mechanism is demonstrated. When a user wishes to run an application on a second machine, but wishes to retain customizations made previously on the first, the client software will handle this by contacting the first machine to retrieve customized files and other customization data. Unmodified files will be retrieved as usual from the application streaming server.

[0205] Here, File 4 exists in three different versions. The server 1503 provides one version of this file 1506, client 11501 has a second version of this file 1504, and client 21502 has a third version 1505. Files may be modified differently for each client.

[0206] The clients may also contain files not present on the server or on other clients. File 51507 is one such file; it exists only on client 11501. File 61508 only exists on client 21502.

[0207] Local Customization

[0208] A local copy-on-write file system allows some applications to write configuration or initialization files where they want to without rewriting the application, and without disturbing the local customization of other clients. Installations of applications on file servers typically do not allow the installation directories of applications to be written, so additional reconfiguration or rewrites of applications are usually necessary to allow per-user customization of some settings.

[0209] With respect to FIG. 16, the cache 1602 with extensions for supporting local file customization is shown. Each block of data in the cache is marked as "clean" 1604 or "dirty" 1605. Pages marked as dirty have been customized by the client 1609, and cannot be removed from the cache 1602 without losing client customization. Pages marked as clean may be purged from the cache 1602, as they can be retrieved again from the server 1603. The index 1601 indicates which pages are clean and dirty. In FIG. 16, clean pages are white, and dirty pages are shaded. File 11606 contains only clean pages, and thus may be entirely evicted from the cache 1602. File 21607 contains only dirty pages, and cannot be removed at all from the cache 1602. File 31608 contains some clean and some dirty pages 1602. The clean pages of File 31608 may be removed from the cache 1602, while the dirty pages must remain.

[0210] Selective Write Protection

[0211] The client streaming software disallows modifications to certain application files. This provides several benefits, such as preventing virus infections and reducing the chance of accidental application corruption. Locally installed files are typically not protected in any way other than conventional backup. Application file servers may be protected against writing by client machines, but are not typically protected against viruses running on the server itself. Most client file systems allow files to be marked as read-only, but it is typically possible to change a file from read-only to read-write. The client application streaming software will not allow any data to be written to files that are marked as not modifiable. Attempts to mark the file as writeable will not be successful.

[0212] Error Detection and Correction

[0213] The client streaming software maintains checksums of application code and data and can repair damaged or deleted files by retrieving another copy from the application streaming server. Traditional application delivery mechanisms do not make any provisions for detecting or correcting corrupted application installs. The user typically detects a corrupt application, and the only solution is to completely reinstall the application. Corrupt application files are detected by the invention automatically, and replacement code or data are invisibly retrieved by the client streaming software without user intervention.

[0214] When a block of code or data is requested by the client operating system, the client application streaming software will compute the checksum of the data block before it is returned to the operating system. If this checksum does not match that stored in the cache, the client will invalidate the cache entry and retrieve a fresh copy of the page from the server.

[0215] File Identifiers

[0216] Applications may be patched or upgraded via a change in the root directory for that application. Application files that are not affected by the patch or upgrade need not be downloaded again. Most existing file systems do not cache files locally.

[0217] Each file has a unique identifier (number). Files that are changed or added in the upgrade are given new identifiers never before used for this application. Files that are unchanged keep the same number. Directories whose contents change are also considered changes. If any file changes, this will cause its parent to change, all the way up to the root directory.

[0218] Upgrade Mechanism

[0219] When the client is informed of an upgrade, it is told of the new root directory. It uses this new root directory to search for files in the application. When retrieving an old file that hasn't changed, it will find the old file identifier, which can be used for the existing files in the cache. In this way, files that do not change can be reused from the cache without downloading them again. For a file that has changed, when the file name is parsed, the client will find a new file number. Because this file number did not exist before the upgrade, the client will not have this file in the cache, and will stream the new file contents when the file is freshly accessed. This way it always gets the newest version of files that change.

[0220] The client application streaming software can be notified of application upgrades by the application streaming server. These upgrades can be marked as mandatory, in which case the client software will force the application to be upgraded.

[0221] The client will contact the application streaming server when it starts the application. At this time, the streaming application server can inform the client of any upgrades. If the upgrade is mandatory, the client will be informed, and it will automatically begin using the upgraded application by using the new root directory.

[0222] Multicast Technique

[0223] A broadcast or multicast medium may be used to efficiently distribute applications from one application streaming server to multiple application streaming clients. Traditional networked application delivery mechanisms usually involve installing application code and data on a central server and having client machines run the application from that server. The multicast mechanism allows a single server to broadcast or multicast the contents of an application to many machines simultaneously. The client machines will receive the application via the broadcast and save it in their local disk cache. The entire application can be distributed to a large number of client machines from a single server very efficiently.

[0224] The multicast network is any communication mechanism that has broadcast or multicast capability. Such media include television and radio broadcasts and IP multicasting on the Internet. Each client that is interested in a particular application may listen to the multicast media for code and data for that application. The code and data are stored in the cache for later use when the application is run.

[0225] These client techniques can be used to distribute data that changes rarely. Application delivery is the most appealing use for these techniques, but they could easily be adopted to distribute other types of slowly changing code and data, such as static databases.

Load Balancing and Fault Tolerance for Streamed Applications

[0226] This section focuses on load balancing (and thereby scalability) and hardware fail over. Throughout this discussion reference should be made to FIG. 17. Load balancing and fault tolerance are addressed in the invention by using a smart client and smart server combination. A preferred embodiment of the invention that implements these features includes three types of servers (described below): app servers; SLM servers; and an ASP Web server. These are organized as follows:

[0227] 1: ASP Web server 1703—This is the Web server that the user goes to for subscribing to applications, creating accounts etc. Compared to the other two types of servers it is characterized by: lowest traffic, fewest number of them, & least likely to go down.

[0228] 2: SLM Servers 1707—subscription license manager servers—These keep track of which user has subscribed to what applications under what license etc. Compared to the other two types of servers it is characterized by: medium traffic, manageable number, and less likely to go down.

[0229] 3: App Servers 1710—These are the servers to which the users go to for application pages. Compared to the other two types of servers it is characterized by: highest traffic, most number of them, most likely to go down either due to hardware failure or application re-configuration.

[0230] Server Lists

[0231] Clients 1704 subscribe and unsubscribe to applications via the ASP Web server 1703. At that point, instead of getting a primary and a secondary server that can perform the job, the ASP Web server 1703 gives them a non-prioritized list of a large number of SLM servers 1706 that can do the job. When the application starts to run, each client contacts the SLM servers 1707, 1708, 1709 and receive its application server list 1705 that can serve the application in question and also receive the access tokens that can be used to validate themselves with the application servers 1710-1715. All access tokens have an expiration time after which they need to be renewed.

[0232] Server Selection

[0233] Having gotten a server list for each type of server 1705, 1706, the client 1704 will decide which specific server to send its request to. In a basic implementation, a server is picked randomly from the list, which will distribute the client's load on the servers very close to evenly. An alternative preferred implementation will do as follows:

[0234] a) Clients will initially pick servers from the list randomly, but they will also keep track of the overall response time they get from each request; and

[0235] b) As each client learns about response times for each server, it can be more intelligent (rather than random) and pick the most responsive server. It is believed that the client is better suited at deciding which server is most responsive because it can keep track of the round trip response time.

[0236] Client-side Hardware Fail Over

[0237] The server selection logic provides hardware failover in the following manner:

[0238] a) If a server does not respond, i.e., times out, the client 1704 will pick another server from its list 1705, 1706 and re-send the request. Since all the servers in the client's server list 1705, 1706 are capable of processing the client's request, there are no questions of incompatibility.

[0239] b) If a SAS client 1704 gets a second time out, i.e., two servers are down, it re-sends the request to multiple servers from its list 1705, 1706 in parallel. This approach staggers the timeouts and reduces the overall delay in processing a request.

[0240] c) In case of a massive hardware failure, all servers in the client's list 1705, 1706 may be down. At this point, the client will use the interfaces to refresh its server list. This is where the three tiers of servers become important:

[0241] 1) If the client 1704 want to refresh its App server list 1705, it will contact an SLM server 1707, 1709 in its list of SLM servers 1706. Again, the same random (SLM) server selection order is

utilized here. Most of the time, this request will be successful and the client will get an updated list of app servers.

[0242] 2) If for some reason all of the SLM servers 1707, 1709 in the client's list 1706 are down, it will contact the ASP Web server 1703 to refresh its SLM server list 1706.

[0243] This 3-tiered approach significantly reduces the impact of a single point of failure—the ASP Web server 1703, effectively making it a fail over of a fail over.

[0244] Server Load Balancing

[0245] In a preferred embodiment of the invention, a server side monitor 1702 keeps track of the overall health and response times for each server request. The Monitor performs this task for all Application and SLM servers. It posts prioritized lists of SLM servers and app servers 1701 that can serve each of the apps in a database shared by the monitor 1702 and all servers. The monitor's algorithm for prioritizing server lists is dominated by the server's response time for each client request. If any servers fail, the monitor 1702 informs the ASP 1703 and removes it from the server list 1701. Note that the server lists 1705, 1706 that the client 1704 maintains are subsets of lists the monitor 1702 maintains in a shared database 1701.

[0246] Since all servers can access the shared database 1701, they know how to 'cut' a list of servers to a client. For example, the client starts to run an SAS application or it wants to refresh its app server list: It will contact an SLM server and the SLM server will access the database 1701 and cut a list of servers that are most responsive (from the server's prospective).

[0247] In this scheme, the server monitor 1702 is keeping track of what it can track the best: how effectively servers are processing client requests (server's response time). It does not track the network propagation delays etc. that can significantly contribute to a client's observed response time.

[0248] ASP Managing Hardware Failovers

[0249] The foregoing approaches provide an opportunity for ASPs to better manage massive scale failures. Specifically, when an ASP 1703 realizes that massive numbers of servers are down, it can allocate additional resource on a temporary basis. The ASP 1703 can update the central database 1701 such that clients will receive only the list that the ASP 1703 knows to be up and running. This includes any temporary resources added to aid the situation. A particular advantage of this approach is that ASP 1703 doesn't need special actions, e.g., emails or phone support, to route clients over to these temporary resources; the transition happens automatically.

[0250] Handling Client Crashes and Client Evictions

[0251] To prevent the same user from running the same application from multiple machines, the SLM servers 1707, 1708, 1709 track what access tokens have been handed to what users. The SAS file system tracks the beginning and end of applications. The user's SAS client software asks for an access token from the SLM servers 1707, 1708, 1709 at the beginning of an application if it already does not have one and it releases the access token when the application ends. The SLM server makes sure that at a given point only one access token has been given to a particular user. In this manner, the user can run the application from multiple

machines, but only from one at a particular time. However, if the user's machine crashes before the access token has been relinquished or if for some reason the ASP 1703 wants to evict a user, the access token granted to the user must be made invalid. To perform this, the SLM server gets the list of application servers 1705 that have been sent to the client 1704 for serving the application and sends a message to those application servers 1710, 1711, 1713, 1714 to stop serving that particular access token. This list is always maintained in the database so that every SLM server can find out what list is held by the user's machine. The application servers before servicing any access token must check with this list to ensure that the access token has not become invalid. Once the access token expires, it can be removed from this list.

Server-side Performance Optimization

[0252] This section describes approaches that can be taken to reduce client-side latency (the time between when an application page is needed and when it is obtained) and improve Application Server scalability (a measure of the number of servers required to support a given population of clients). The former directly affects the perceived performance of an application by an end user (for application features that are not present in the user's cache), while the latter directly affects the cost of providing application streaming services to a large number of users.

[0253] Application Server Operation

[0254] The basic purpose of the Application Server is to return Application File Pages over the network as requested by a client. The Application Server holds a group of Stream Application Sets from which it obtains the Application File Pages that match a client request. The Application Server is analogous to a typical network file system (which also returns file data), except it is optimized for delivery of Application file data, i.e., code or data that belong directly to the application, produced by the software provider, as opposed to general user file data (document files and other content produced by the users themselves). The primary differences between the Application Server and a typical network file system are:

- [0255] 1. The restriction to handle only Application file data allows the Application Server to only service read requests, with writes being disallowed or handled on the client itself in a copy-on-write manner;
- [0256] 2. Access checks occur at the application level, that is a client is given all-or-none access to files for a given software application;
- [0257] 3. The Application Server is designed to operate across the Internet, as opposed to typical network file systems, which are optimized to run over LANs. This brings up additional requirements of handling server failures, maximizing network bandwidth and minimizing latency, and handling security; and
- [0258] 4. The Application Server is application-aware, unlike typical network file systems, which treat all software application files the same as all other files. This allows the Application Server to use and collect per-application access profile information along with other statistics.

[0259] To service a client request, the Application Server software component keeps master copies of the full Application Stream Sets on locally accessible persistent storage. In main memory, the Application Server maintains a cache of commonly accessed Application File Pages. The primary steps taken by the Application Server to service a client request are:

- [0260] 1. Receive and decode the client request;
- [0261] 2. Validate the client's privilege to access the requested data, e.g., by means of a Kerberos-style ticket issued by a trusted security service;
- [0262] 3. Look up the requested data in the main memory cache, and failing that, obtain it from the master copy on disk while placing it in the cache; and
- [0263] 4. Return the File Pages to the client over the network.

[0264] The techniques used to reduce latency and improve server scalability (the main performance considerations) are described below.

[0265] Server Optimization Features

[0266] Read-Only File System for Application Files—Because virtually all application files (code and data) are never written to by users, virtually the entire population of users have identical copies of the application files. Thus a system intending to deliver the application files can distribute a single, fixed image across all servers. The read-only file system presented by the Application Server represents this sharing, and eliminates the complexities of replication management, e.g., coherency, that occur with traditional network file systems. This simplification enables the Application Servers to respond to requests more quickly, enables potential caching at intervening nodes or sharing of caches across clients in a peer-to-peer fashion, and facilitates fail over, since with the read-only file system the Application File Pages as identified by the client (by a set of unique numbers) will always globally refer to the same content in all cases.

[0267] Per-page Compression—Overall latency observed by the client can be reduced under low-bandwidth conditions by compressing each Application File Page before sending it. Referring to FIG. 18, the benefits of the use of compression in the streaming of Application File Pages, is illustrated. The client 1801 and server 1802 timelines are shown for a typical transfer of data versus the same data sent in a compressed form. The client requests the data from the server 1803. The server processes the request 1804 and begins sending the requested data. The timelines then diverge due to the ability to stream the compressed data 1805 faster than the uncompressed data 1806.

[0268] With respect to FIG. 19, the invention's pre-compression of Application File Pages process is shown. The Builder generates the stream application sets 1901, 1902 which are then pre-compressed by the Stream Application Set Post-Processor 1903. The Stream Application Set Post-Processor 1903 stores the compressed application sets in the persistent storage device 1904. Any client requests for data are serviced by the Application Server which sends the pre-compressed data to the requesting client 1905. The reduction in size of the data transmitted over the network reduces the time to arrival (though at the cost of some processing time on the client to decompress the data). When

the bandwidth is low relative to processing power, e.g., 256 kbps with a Pentium-III-600, this can reduce latency significantly.

[0269] **Page-set Compression**—When pages are relatively small, matching the typical virtual memory page size of 4 kB, adaptive compression algorithms cannot deliver the same compression ratios that they can for larger blocks of data, e.g., 32 kB or larger. Referring to FIG. 20, when a client 2001 requests multiple Application File Pages at one time 2002, the Application Server 2006 can concatenate all the requested pages and compress the entire set at once 2004, thereby further reducing the latency the client will experience due to the improved compression ratio. If the pages have already been compressed 2003, then the request is fulfilled from the cache 2007 where the compressed pages are stored. The server 2006 responds to the client's request through the transfer of the compressed pages 2005.

[0270] **Post-processing of Stream Application Sets**—The Application Server may want to perform some post processing of the raw Stream Application Sets in order to reduce its runtime-processing load, thereby improving its performance. One example is to pre-compress all Application File Pages contained in the Stream Application Sets, saving a great deal of otherwise repetitive processing time. Another possibility is to rearrange the format to suit the hardware and operating system features, or to reorder the pages to take advantage of access locality.

[0271] **Static and Dynamic Profiling**—With respect to FIG. 21, since the same application code is executed in conjunction with a particular Stream Application Set 2103 each time, there will be a high degree of temporal locality of referenced Application File Pages, e.g., when a certain feature is invoked, most if not all the same code and data is referenced each time to perform the operation. These access patterns can be collected into profiles 2108, which can be shipped to the client 2106 to guide its prefetching (or to guide server-based 2105 prefetching), and they can be used to pre-package groups of Application File Pages 2103, 2104 together and compress them offline as part of a post-processing step 2101, 2102, 2103. The benefit of the latter is that a high compression ratio can be obtained to reduce client latency without the cost of runtime server processing load (though only limited groups of Application File Pages will be available, so requests which don't match the profile would get a superset of their request in terms of the pre-compressed groups of Application File Pages that are available).

[0272] **Fast Server-Side Client Privilege Checks**—Referring to FIG. 22, having to track individual user's credentials, i.e., which Applications they have privileges to access, can limit server scalability since ultimately the per-user data must be backed by a database, which can add latency to servicing of user requests and can become a central bottleneck. Instead, a separate License Server 2205 is used to offload per-user operations to grant privileges to access application data, and thereby allow the two types of servers 2205, 2210 to scale independently. The License Server 2205 provides the client an Access Token (similar to a Kerberos ticket) that contains information about what application it represents rights for along with an expiration time. This simplifies the operations required by the Application Server 2210 to validate a client's privileges 2212. The Application

Server 2210 needs only to decrypt the Access Token (or a digest of it) via a secret key shared 2209 with the License Server 2205 (thus verifying the Token is valid), then checking the validity of its contents, e.g., application identifier, and testing the expiration time. Clients 2212 presenting Tokens for which all checks pass are granted access. The Application Server 2210 needs not track anything about individual users or their identities, thus not requiring any database operations. To reduce the cost of privilege checks further, the Application Server 2210 can keep a list of recently used Access Tokens for which the checks passed, and if a client passes in a matching Access Token, the server need only check the expiration time, with no further decryption processing required.

[0273] **Connection Management**—Before data is ever transferred from a client to a server, the network connection itself takes up one and a half network round trips. This latency can adversely impact client performance if it occurs for every client request. To avoid this, clients can use a protocol such as HTTP 1.1, which uses persistent connections, i.e., connections stay open for multiple requests, reducing the effective connection overhead. Since the client-side file system has no knowledge of the request patterns, it will simply keep the connection open as long as possible. However, because traffic from clients may be bursty, the Application Server may have more open connections than the operating system can support, many of them being temporarily idle. To manage this, the Application Server can aggressively close connections that have been idle for a period of time, thereby achieving a compromise between the client's latency needs and the Application Server's resource constraints. Traditional network file systems do not manage connections in this manner, as LAN latencies are not high enough to be of concern.

[0274] **Application Server Memory Usage/Load Balancing**—File servers are heavily dependent on main memory for fast access to file data (orders of magnitude faster than disk accesses). Traditional file servers manage their main memory as cache of file blocks, keeping the most commonly accessed ones. With the Application Server, the problem of managing main memory efficiently becomes more complicated due to there being multiple servers providing a shared set of applications. In this case, if each server managed its memory independently, and was symmetric with the others, then each server would only keep those file blocks most common to all clients, across all applications. This would cause the most common file blocks to be in the main memory of each and every Application server, and since each server would have roughly the same contents in memory, adding more servers won't improve scalability by much, since not much more data will be present in memory for fast access. For example, if there are application A (accessed 50% of the time), application B (accessed 40% of the time), and application C (accessed 10% of the time), and application A and B together consume more memory cache than a single Application Server has, and there are ten Application Servers, then none of the Application Servers will have many blocks from C in memory, penalizing that application, and doubling the number of servers will improve C's performance only minimally. This can be improved upon by making the Application Servers asymmetric, in that a central mechanism, e.g., system administrator, assigns individual Application Servers different Application Stream Sets to provide, in accordance with

popularity of the various applications. Thus, in the above example, of the ten servers, five can be dedicated to provide A, four to B, and one to C, (any extra memory available for any application) making a much more effective use of the entire memory of the system to satisfy the actual needs of clients. This can be taken a step further by dynamically (and automatically) changing the assignments of the servers to match client accesses over time, as groups of users come and go during different time periods and as applications are added and removed from the system. This can be accomplished by having servers summarize their access patterns, send them to a central control server, which then can reassign servers as appropriate.

Conversion of Conventional Applications to Enable Streamed Delivery and Execution

[0275] The Streamed Application Set Builder is a software program. It is used to convert locally installable applications into a data set suitable for streaming over a network. The streaming-enabled data set is called the Streamed Application Set (SAS). This section describes the procedure used to convert locally installable applications into the SAS.

[0276] The application conversion procedure into the SAS consists of several phases. In the first phase, the Builder program monitors the installation process of a local installation of the desired application for conversion. The Builder monitors any changes to the system and records those changes in an intermediate data structure. After the application is installed locally, the Builder enters the second phase of the conversion. In the second phase, the Builder program invokes the installed application executable and obtains sequences of frequently accessed file blocks of this application. Both the Builder program and the client software use the sequence data to optimize the performance of the streaming process. Once the sequencing information is obtained, the Builder enters the final phase of the conversion. In this phase, the Builder gathers all data obtained from the first two phases and processes the data into the Streamed Application Set.

[0277] Detailed descriptions of the three phases of the Builder conversion process are described in the following sections. The three phases consist of installation monitoring (IM), application profiling (AP), and SAS packaging (SP). In most cases, the conversion process is general and applicable to all types of systems. In places where the conversion is OS dependent, the discussion is focused on the Microsoft Windows environment. Issues on conversion procedure for other OS environments are described in later sections.

[0278] Installation Monitoring (IM)

[0279] In the first phase of the conversion process, the Builder Installation Monitor (IM) component invokes the application installation program that installs the application locally. The IM observes all changes to the local computer during the installation. The changes may involve one or more of the following: changes to system or environment variables; and modifications, addition, or deletion of one or more files. Initial system variables, environment variables, and files are accounted for by the IM before the installation begins to give a more accurate picture of any changes that are observed. The IM records all changes to the variables and files in a data structure to be sent to the Builder's Streamed Application Packaging component. In the following paragraphs, detailed description of the Installation Monitor is described for Microsoft Windows environment.

[0280] In Microsoft Windows system, the Installation Monitor (IM) component consists of a kernel-mode driver subcomponent and a user-mode subcomponent. The kernel-mode driver is hooked into the system registry and file system function interface calls. The hook into the registry function calls allows the IM to monitor system variable changes. The hook into the file system function calls enables the IM to observe file changes.

[0281] Installation Monitor Kernel-Mode subcomponent (IM-KM)

[0282] With respect to FIG. 23, the IM-KM subcomponent monitors two classes of information during an application installation: system registry modifications and file modifications. Different techniques are used for each of these classes.

[0283] To monitor system registry modifications 2314, the IM-KM component replaces all kernel-mode API calls in the System Service Table that write to the system registry with new functions defined in the IM-KM subcomponent. When an installation program calls one of the API functions to write to the registry 2315, the IM-KM function is called instead, which logs the modification data 2317 (including registry key path, value name and value data) and then forwards the call to the actual operating system defined function 2318. The modification data is made available to the IM-UM subcomponent through a mechanism described below.

[0284] To monitor file modifications, a filter driver is attached to the file system's driver stack. Each time an installation program modifies a file on the system, a function is called in the IM-KM subcomponent, which logs the modification data (including file path and name) and makes it available to the IM-UM using a mechanism described below.

[0285] The mechanisms used for monitoring registry modifications and file modifications will capture modifications made by any of the processes currently active on the computer system. While the installation program is running, other processes that, for example, operate the desktop and service network connections may be running and may also modify files or registry data during the installation. This data must be removed from the modification data to avoid inclusion of modifications that are not part of the application installation. The IM-KM uses process monitoring to perform this filtering.

[0286] To do process monitoring, the IM-KM installs a process notification callback function that is called each time a process is created or destroyed by the operating system. Using this callback function, the operating system sends the created process ID as well as the process ID of the creator (or parent) process. The IM-KM uses this information, along with the process ID of the IM-UM, to create a list of all of the processes created during the application installation. The IM-KM uses the following algorithm to create this list:

[0287] 1. Before the installation program is launched by the IM-UM, the IM-UM passes its own process ID to the IM-KM. Since the IM-UM is launching the installation application, the IM-UM will be the ancestor (parent, grandparent, etc.) of any process (with one exception—the Installer Service described below) that modifies files or registry data as part of the application installation.

[0288] 2. When the installation is launched and begins the creating processes, the IM-KM process monitoring logic is notified by the operating system via the process notification callback function.

[0289] 3. If the creator (parent) process ID sent to the process notification callback function is already in the process list, the new process is included in the list.

[0290] When an application on the system modifies either the registry or files, and the IM-KM monitoring logic captures the modification data, but before making it available to the IM-UM, it first checks to see if the process that modified the registry or file is part of the process list. It is only made available to the IM-UM if it is in the process list.

[0291] It is possible that a process that is not a process ancestor of the IM-UM will make changes to the system as a proxy for the installation application. Using interprocess communication, an installation program may request than an Installer Service make changes to the machine. In order for the IM-KM to capture changes made by the Installer Service, the process monitoring logic includes a simple rule that also includes any registry or file changes that have been made by a process with the same name as the Installer Service process. On Windows 2000, for example, the Installer Service is called "msi.exe".

[0292] Installation Monitor User-Mode subcomponent (IM-UM)

[0293] The IM kernel-mode (IM-KM) driver subcomponent is controlled by the user-mode subcomponent (IM-UM). The IM-UM sends messages to the IM-KM to start 2305 and stop 2309 the monitoring process via standard I/O control messages known as IOCTLs. The message that starts the IM-KM also passes in the process ID of the IM-UM to facilitate process monitoring described in the IM-KM description.

[0294] When the installation program 2306 modifies the computer system, the IM-KM signals a named kernel event. The IM-UM listens for these events during installation. When one of these events is signaled, the IM-UM calls the IM-KM using an IOCTL message. In response, the IM-KM packages data describing the modification and sends it to the IM-UM 2318.

[0295] The IM-UM sorts this data and removes duplicates. Also, it parameterizes all local-system-specific registry keys, value names, and values. For example, an application will often store paths in the registry that allow it to find certain files at run-time. These path specifications must be replaced with parameters that can be recognized by the client installation software.

[0296] A user interface is provided for the IM-UM that allows an operator of the Builder to browse through the changes made to the machine and to edit the modification data before the data is packaged into an SAS.

[0297] Once the installation of an application is completed 2308, the IM-UM forwards data structures representing the file and registry modifications to the Streamed Application Packager 2312.

[0298] Monitoring Application Configuration

[0299] Using the techniques described above for monitoring file modifications and monitoring registry modifications, the builder can also monitor a running application that is being configured for a particular working environment. The data acquired by the IM-UM can be used to duplicate the same configuration on multiple machines, making it unnecessary for each user to configure his/her own application installation.

[0300] An example of this is a client server application for which the client will be streamed to the client computer system. Common configuration modifications can be captured by the IM and packed into the SAS. When the application is streamed to the client machine, it is already configured to attach to the server and begin operation.

[0301] Application Profiling (AP)

[0302] Referring to FIG. 24, in the second phase of the conversion process, the Builder's Application Profiler (AP) component invokes the application executable program that is installed during the first phase of the conversion process. Given a particular user input, the executable program file blocks are accessed in a particular sequence. The purpose of the AP is to capture the sequence data associated with some user inputs. This data is useful in several ways.

[0303] First of all, frequently used file blocks can be streamed to the client machine before other less used file blocks. A frequently used file block is cached locally on the client cache before the user starts using the streamed application for the first time. This has the effect of making the streamed application as responsive to the user as the locally installed application by hiding any long network latency and bandwidth problems.

[0304] Secondly, the frequently accessed files can be reordered in the directory to allow faster lookup of the file information. This optimization is useful for directories with large number of files. When the client machine looks up a frequently used file in a directory, it finds this file early in the directory search. In an application run with many directory queries, the performance gain is significant.

[0305] Finally, the association of a set of file blocks with a particular user input allows the client machine to request minimum amount of data needed to respond to that particular user command. The profile data association with a user command is sent from the server to the client machine in the AppInstallBlock during the 'preparation' of the client machine for streaming. When the user on a client machine invokes a particular command, the codes corresponding to this command are prefetched from the server.

[0306] The Application Profiler (AP) is not as tied to the system as the Installation Monitor (IM) but there are still some OS dependent issues. In the Windows system, the AP still has two subcomponents: kernel-mode (AP-KM) subcomponent and the user-mode (AP-UM) subcomponent. The AP-UM invokes the converting application executable. Then AP-UM starts the AP-KM 2403, 2413 to track the sequences of file block accesses by the application 2414. Finally when the application exits after the pre-specified amount of sequence data is gathered, the AP-UM retrieves the data from AP-KM 2406, 2417 and forwards the data to the Streamed Application Packager 2411.

[0307] Streamed Application Set Packaging (SP)

[0308] With respect to **FIG. 25**, in the final phase of the conversion process, the Builder's Streamed Application Set Packager (SP) component processes the data structure from IM and AP to create a data set suitable for streaming over the network. This converted data set is called the Streamed Application Set **2520** and is suitable for uploading to the Streamed Application Servers for subsequent downloading by the stream client. **FIG. 23** shows the control flow of the SP module.

[0309] Each file included in a Streamed Application Set **2520** is assigned a file number that identifies it within the SAS.

[0310] The Streamed Application Set **2520** consists of the three sets of data from the Streamed Application Server's perspective. The three types of data are the Concatenation Application File (CAF) **2519**, **2515**, the Size Offset File Table (SOFT) **2518**, **2514**, **2507**, and the Root Versioning Table (RVT) **2518**, **2514**.

[0311] The CAF **2519**, **2515** consists of all the files and directories needed to stream to the client. The CAF can be further divided into two subsets: initialization data set and the runtime data set.

[0312] The initialization data set is the first set of data to be streamed from the server to the client. This data set contains the information captured by IM and AP needed by the client to prepare the client machine for streaming this particular application. This initialization data set is also called the AppInstallBlock (AIB) **2516**, **2512**. In addition to the data captured by the IM and AP modules, the SP is also responsible for merging any new dynamic profile data gathered from the client and the server. This data is merged into the existing AppInstallBlock to optimize subsequent streaming of the application **2506**. With the list of files obtained by the IM during application installation, the SP module separates the list of files into regular streamed files and the spoof files. The spoof files consists of those files not installed into standard application directory. This includes files installed into system directories and user specific directories. The detailed format description of the AppInstallBlock is described later.

[0313] The second part of the CAF consists of the runtime data set. This is the rest of the data that is streamed to the client once the client machine is initialized for this particular application. The runtime data consists of all the regular application files and the directories containing information about those application files. Detailed format description of the runtime data in the CAF section is described below. The SP appends every file recorded by IM into the CAF and generates all directories. Each directory contains list of file name, file number, and the metadata associated with the files in that particular directory **2517**, **2513**.

[0314] The SP is also responsible for generating the SOFT file **2518**, **2514**, **2507**. This is a table used to index into the CAF for determining the start and the end of a file. The server uses this information to quickly access the proper file within the directory for serving the proper file blocks to the client.

[0315] Finally, the SP creates the RVT file **2518**, **2514**. The Root Versioning Table contains a list of root file number and version number. This information is used to track minor application patches and upgrades. Each entry in the RVT corresponds to one patch level of the application with a

corresponding new root directory. The SP generates new parent directories when any single file in that subdirectory tree is changed from the patched upgrade. The RVT is uploaded to the server and requested by the client at appropriate time for the most updated version of the application by a simple comparison of the client's Streamed Application root file number with the RVT table located on the server once the client is granted access authorization to retrieve the data.

[0316] With respect to **FIGS. 26a** and **26b**, the internal representation of a simple SAS before and after a new file is added to a new version of an application is shown. The original CAF **2601** has the new files **2607** appended to it **2604** by the SP. The SOFT **2602** is correspondingly updated **2605** with the appropriate table entries **2608** to index the new files **2607** the CAF **2604**. Finally, the RVT **2603** is updated **2606** to reflect the new version **2609**.

[0317] Data Flow Description

[0318] The following list describes the data that is passed from one component to another. The numbers corresponds to the numbering in the Data Flow diagram of **FIG. 27**.

[0319] Install Monitor

[0320] 1. The full pathname of the installer program is queried from the user by the Builder program and is sent to the Install Monitor.

[0321] 2. The Install Monitor (IM) user-mode sends a read request to the OS to spawn a new process for installing the application on the local machine.

[0322] 3. The OS loads the application installer program into memory and runs the application installer program. OS returns the process ID to the IM.

[0323] 4. The application program is started by the IM-UM.

[0324] 5. The application installer program sends read request to the OS to read the content of the CD.

[0325] 6. The CD media data files are read from the CD.

[0326] 7. The files are written to the appropriate location on the local hard-drive.

[0327] 8. IM kernel-mode captures all file read/write requests and all registry read/write requests by the application installer program.

[0328] 9. IM user-mode program starts the IM kernel-mode program and sends the request to start capturing all relevant file and registry data.

[0329] 10. IM kernel-mode program sends the list of all file modifications, additions, and deletions; and all registry modifications, additions, and deletions to the IM user-mode program.

[0330] 11. IM informs the SAS Builder UI that the installation monitoring has completed and displays the file and registry data in a graphical user interface.

[0331] Application Profiler

[0332] 12. Builder UI invokes Application Profiling (AP) user-mode program by querying the user for the list of application executable names to be profiled.

The AP user-mode also queries the user for division of file blocks into sections corresponding to the commands invoked by the user of the application being profiled.

- [0333] 13. Application Profiler user-mode invokes each application executable in succession by spawning each program in a new process. The OS loads the application executable into memory, runs the application executable, and returns the process ID to the Application Profiler.
- [0334] 14. During execution, the OS on behalf of the application, sends the request to the hard-drive controller to read the appropriate file blocks into memory as needed by the application.
- [0335] 15. The hard-drive controller returns all file blocks requested by the OS.
- [0336] 16. Every file access to load the application file blocks into memory is monitored by the Application Profiler (AP) kernel-mode program.
- [0337] 17. The AP user-mode program informs the AP kernel-mode program to start monitoring relevant file accesses.
- [0338] 18. Application Profiler kernel-mode returns the file access sequence and frequency information to the user-mode program.
- [0339] 19. Application Profiler returns the processed profile information. This has two sections. The first section is used to identify the frequency of files accessed. The second section is used to list the file blocks for prefetch to the client. The file blocks can be further categorized into subsections according to the commands invoked by the user of the application.
- [0340] SAS Packager
 - [0341] 20. The Streamed Application Packager receives files and registry changes from the Builder UI. It also receives the file access frequency and a list of file blocks from the Profiler. File numbers are assigned to each file.
 - [0342] 21. The Streamed Application Packager reads all the file data from the hard-drive that are copied there by the application installer.
 - [0343] 22. The Streamed Application Packager also reads the previous version of Streamed Application Set for support of minor patch upgrades.
 - [0344] 23. Finally, the new Streamed Application Set data is stored back to non-volatile storage.
 - [0345] 24. For new profile data gathered after the SAS has been created, the packager is invoked to update the AppInstallBlock in the SAS with the new profile information.
- [0346] Mapping of Data Flow to Streamed Application Set (SAS)
 - [0347] Step 7: Data gathered from this step consist of the registry and file modification, addition, and deletion. The data are mapped to the AppInstallBlock's File Section, Add Registry Section, and Remove Registry Section.

[0348] Step 8 & 19: File data are copied to the local hard-drive then concatenated into part of the CAF contents. Part of the data is identified as spoof or copied files and the file names and/or contents are added to the AppInstallBlock.

[0349] Step 15 & 21: Part of the data gathered by the Profiler or gathered dynamically by the client is used in the AppInstallBlock as a prefetch hint to the client. Another part of the data is used to generate a more efficient SAS Directory content by ordering the files according the usage frequency.

[0350] Step 20: If the installation program was an upgrade, SAS Packager needs previous version of the Streamed Application Set data. Appropriate data from the previous version are combined with the new data to form the new Streamed Application Set.

[0351] Format of Streamed Application Set

[0352] Referring to FIG. 28, the format of the Streamed Application Set consists of three sections: Root Version Table (RVT) 2802, Size Offset File Table (SOFT) 2803, and Concatenation Application File (CAF) 2801. The RVT section 2802 lists all versions of the root file numbers available in a Streamed Application Set. The SOFT 2803 section consists of the pointers into the CAF 2801 section for every file in the CAF 2801. The CAF section 2801 contains the concatenation of all the files associated with the streamed application. The CAF section 2801 is made up of regular application files, SAS directory files 2805, AppInstallBlock 2804, and icon files. See below for detailed information on the content of the SAS file.

[0353] OS Dependent Format

[0354] The format of the Streamed Application Set is designed to be as portable as possible across all OS platforms. At the highest level, the format of CAF, SOFT, and RVT that make up the format of the Streamed Application Set are completely portable across any OS platforms. One piece of data structure that is OS dependent is located in the initialization data set called AppInstallBlock in the CAF. This data is dependent on the type of OS due to the differences in low-level system differences among different OS. For example, Microsoft Windows contains system environment variables called the Registry. The Registry has a particular tree format not found in other operating systems like UNIX or MacOS.

[0355] Another OS dependent piece of data is located in the SAS directory files in the CAF. The directory contains file metadata information specific to Windows files. For example on the UNIX platform, there does not exist a hidden flag. This platform specific information needs to be transmitted to the client to fool the streamed application into believing that the application data is located natively on the client machine with all the associated file metadata intact. If SAS is to be used to support streaming of UNIX or MacOS applications, file metadata specific to those systems will need to be recorded in the SAS directory.

[0356] Lastly, the format of the file names itself is OS dependent. Applications running on the Windows environment inherit the old MSDOS 8.3 file name format. To support this properly, the format of the SAS Directory file in

CAF requires an additional 8.3 field to store this information. This field is not needed in other operating systems like UNIX or MacOS.

[0357] Device Driver Versus File System Paradigm

[0358] Referring to **FIGS. 29 and 30**, the SAS client Prototype is implemented using the 'device driver' paradigm. One of the advantages of the device driver approach is that the caching of the sector blocks is simpler. In the device driver approach, the client cache manager **2902** only needs to track sector numbers in its cache **2903**. In comparison with the 'file system' paradigm, more complex data structure are required by the client cache manager **3002** to track a subset of a file that is cached **3003** on a client machine. This makes the device driver paradigm easier to implement.

[0359] On the other hand, there are many drawbacks to the device driver paradigm. On the Windows system, the device driver approach has a problem supporting large numbers of applications. This is due to the phantom limitation on the number of assignable drive letters available in a Windows system (26 letters); and the fact that each application needs to be located on its own device. Note that having multiple applications on a device is possible, but then the server needs to maintain an exponential number of devices that support all possible combinations of applications. This is too costly to maintain on the server.

[0360] Another problem with the device driver approach is that the device driver operates at the disk sector level. This is a much lower level than operating at the file level in the file system approach. The device driver does not know anything about files. Thus, the device driver cannot easily interact with the file level issues. For example, spoofing files and interacting with the OS file cache is nearly impossible with the device driver approach. Both spoofing files and interacting with the OS buffer cache are needed to get higher performance. In addition, operating at the file system level leads to optimizing the file system to better suit this approach of running applications. For instance, typical file systems do logging and make multiple disk sector requests at a time. These are not needed in this approach and are actually detrimental to the performance. When operating at the device driver level, not much can be done about that. Also, operating at the file system level helps in optimizing the protocol between the client and the server.

[0361] Implementation in the Prototype

[0362] The prototype has been implemented and tested successfully on the Windows and Linux distributed system. The prototype is implemented using the device driver paradigm as described above. The exact procedure for streaming application data is described next.

[0363] First of all, the prototype server is started on either the Windows-based or Linux-based system. The server creates a large local file mimicking large local disk images. Once the disk images are prepared, it listens to TCP/IP ports for any disk sector read or write requests.

[0364] Implementation of SAS Builder

[0365] The SAS Builder has been implemented on the Windows-based platform. A preliminary Streamed Application Set file has been created for real-world applications like Adobe Photoshop. A simple extractor program has been

developed to extract the SAS data on a pristine machine without the application installed locally. Once the extractor program is run on the SAS, the application runs as if it was installed locally on that machine. This process verifies the correctness of the SAS Building process.

Format of Streamed Application Set (SAS)

[0366] Functionality

[0367] The streamed application set (SAS), illustrated in **FIG. 28**, is a data set associated with an application suitable for streaming over the network. The SAS is generated by the SAS Builder program. The program converts locally installable applications into the SAS. This section describes the format of the SAS.

[0368] Note: Fields greater than a single byte are stored in little-endian format. The Stream Application Set (SAS) file size is limited to 2^{64} bytes. The files in the CAF section are laid out in the same order as its corresponding entries in the SOFT table.

[0369] Data Type Definitions

[0370] The format of the SAS consists of four sections: header, Root Version Table (RVT), Size Offset File Table (SOFT), and Concatenation Application File (CAF) sections.

[0371] 1. Header Section

[0372] MagicNumber [4 bytes]: Magic number identifying the file content with the SAS.

[0373] ESSVersion [4 bytes]: Version number of the SAS file format.

[0374] AppID [16 bytes]: A unique application ID for this application. This field must match the AppID located in the AppInstallBlock. Window Guidgen API is used to create this identifier.

[0375] Flags [4 bytes]: Flags pertaining to SAS.

[0376] Reserved [32 bytes]: Reserved spaces for future.

[0377] RVTOffset [8 bytes]: Byte offset into the start of the RVT section.

[0378] RVTSIZE [8 bytes]: Byte size of the RVT section.

[0379] SOFTOffset [8 bytes]: Byte offset into the start of the SOFT section.

[0380] SOFTSIZE [8 bytes]: Byte size of the SOFT section.

[0381] CAFOffset [8 bytes]: Byte offset into the start of the CAF section.

[0382] CAFSIZE [8 bytes]: Byte size of the CAF section.

[0383] VendorNameIsAnsi [1 byte]: 0 if the vendor name is in Unicode format. 1 if the vendor name is in ANSI format.

[0384] VendorNameLength [4 bytes]: Byte length of the vendor name.

- [0385] VendorName [X bytes]: Name of the software vendor who created this application. e.g., "Microsoft". Null-terminated.
- [0386] AppBaseNameIsAnsi [1 byte]: 0 if the vendor name is in Unicode format. 1 if the vendor name is in ANSI format.
- [0387] AppBaseNameLength [4 bytes]: Byte length of the application base name.
- [0388] AppBaseName [X bytes]: Base name of the application. e.g., "Word 2000". Null-terminated.
- [0389] MessageIsAnsi [1 byte]: 0 if the vendor name is in Unicode format. 1 if the vendor name is in ANSI format.
- [0390] MessageLength [4 bytes]: Byte length of the message text.
- [0391] Message [X bytes]: Message text. Null-terminated.
- [0392] 2. Root Version Table (RVT) Section
- [0393] The Root version entries are ordered in a decreasing value according to their file numbers. The Builder generates unique file numbers within each SAS in a monotonically increasing value. So larger root file numbers imply later versions of the same application. The latest root version is located at the top of the section to allow the SAS Server easy access to the data associated with the latest root version.
- [0394] NumberEntries [4 bytes]: Number of patch versions contained in this SAS. The number indicates the number of entries in the Root Version Table (RVT).
- [0395] Root Version structure: (variable number of entries)
- [0396] VersionNumber [4 bytes]: Version number of the root directory.
- [0397] FileNumber [4 bytes]: File number of the root directory.
- [0398] VersionNameIsAnsi [1 byte]: 0 if the vendor name is in Unicode format. 1 if the vendor name is in ANSI format.
- [0399] VersionNameLength [4 bytes]: Byte length of the version name
- [0400] VersionName [X bytes]: Application version name. e.g., "SP 1".
- [0401] Metadata [32 bytes]: See SAS FS Directory for format of the metadata.
- [0402] 3. Size Offset File Table (SOFT) Section
- [0403] The SOFT table contains information to locate specific files in the CAF section. The entries are ordered according to the file number starting from 0 to NumberFiles-1. The start of the SOFT table is aligned to eight-byte boundaries for faster access.
- [0404] SOFT entry structure: (variable number of entries)
- [0405] Offset [8 bytes]: Byte offset into CAF of the start of this file.
- [0406] Size [8 bytes]: Byte size of this file. The file is located from address Offset to Offset+Size.
- [0407] 4. Concatenation Application File (CAF) Section
- [0408] CAF is a concatenation of all file or directory data into a single data structure. Each piece of data can be a regular file, an AppInstallBlock, an SAS FS directory file, or an icon file.
- [0409] a. Regular Files
- [0410] FileData [X bytes]: Content of a regular file
- [0411] b. AppInstallBlock (See AppInstallBlock section for detailed format) A simplified description of the AppInstallBlock is listed here. The exact detail of the individual fields in the AppInstallBlock are described later.
- [0412] Header section [X bytes]: Header for AppInstallBlock containing information to identify this AppInstallBlock.
- [0413] Files section [X bytes]: Section containing file to be copied or spoofed.
- [0414] AddVariable section [X bytes]: Section containing system variables to be added.
- [0415] RemoveVariable section [X bytes]: Section containing system variables to be removed.
- [0416] Prefetch section [X bytes]: Section containing pointers to file blocks to be prefetched to the client.
- [0417] Profile section [X bytes]: Section containing profile data.
- [0418] Comment section [X bytes]: Section containing comments about AppInstallBlock.
- [0419] Code section [X bytes]: Section containing application-specific code needed to prepare local machine for streaming this application
- [0420] LicenseAgreement section [X bytes]: Section containing licensing agreement message.
- [0421] c. SAS Directory
- [0422] An SAS Directory contains information about the subdirectories and files located within this directory. This information is used to store metadata information related to the files associated with the streamed application. This data is used to fool the application into thinking that it is running locally on a machine when most of the data is resided elsewhere.
- [0423] The SAS directory contains information about files in its directory. The information includes file number, names, and metadata associated with the files.
- [0424] MagicNumber [4 bytes]: Magic number for SAS directory file.
- [0425] ParentFileID [16+4 bytes]: AppID+FileNumber of the parent directory. AppID is set to 0 if the directory is the root.
- [0426] SelfFileID [16+4 bytes]: AppID+FileNumber of this directory.
- [0427] NumFiles [4 bytes]: Number of files in the directory.
- [0428] Variable-Sized File Entry:
- [0429] UsedFlag [1 byte]: 1 for used, 0 for unused.
- [0430] ShortLen [1 byte]: Length of short file name.

- [0431] LongLen [2 byte]: Length of long file name.
- [0432] NameHash [4 bytes]: Hash value of the short file name for quick lookup without comparing whole string.
- [0433] ShortName [24 bytes]: 8.3 short file name in Unicode. Not null-terminated.
- [0434] FileID [16+4 bytes]: AppID+FileNumber of each file in this directory.
- [0435] Metadata [32 bytes]: The metadata consists of file byte size (8 bytes), file creation time (8 bytes), file modified time (8 bytes), attribute flags (4 bytes), SAS flags (4 bytes). The bits of the attribute flags have the following meaning:
 - [0436] Bit 0: Read-only—Set if file is read-only
 - [0437] Bit 1: Hidden—Set if file is hidden from user
 - [0438] Bit 2: Directory—Set if the file is an SAS Directory
 - [0439] Bit 3: Archive—Set if the file is an archive
 - [0440] Bit 4: Normal—Set if the file is normal
 - [0441] Bit 5: System—Set if the file is a system file
 - [0442] Bit 6: Temporary—Set if the file is temporary
- [0443] The bits of the SAS flags have the following meaning:
 - [0444] Bit 0: ForceUpgrade—Used only on root file. Set if client is forced to upgrade to this particular version if the current root version on the client is older.
 - [0445] Bit 1: RequireAccessToken—Set if file require access token before client can read it.
 - [0446] Bit 2: Read-only—Set if the file is read-only
- [0447] LongName [X bytes]: Long filename in Unicode format with null-termination character.
- [0448] d. Icon files
 - [0449] IconFileData [X bytes]: Content of an icon file.

Format of AppInstallBlock

- [0450] Functionality
- [0451] With respect to FIGS. 31a-31h, the AppInstallBlock is a block of code and data associated with a particular application. This AppInstallBlock contains the information needed to by the SAS client to initialize the client machine before the streamed application is used for the first time. It also contains optional profiling data for increasing the runtime performance of that streamed application.
- [0452] The AppInstallBlock is created offline by the SAS Builder program. First of all, the Builder monitors the installation process of a local version of the application installation program and records changes to the system. This includes any environment variables added or removed from the system 3103, and any files added or modified in the

system directories 3102. Files added to the application specific directory are not recorded in the AppInstallBlock to reduce the amount of time needed to send the AppInstallBlock to the SAS client. Secondly, the Builder profiles the application to obtain the list of critical pages needed to run the application initially and an initial page reference sequence of the pages accessed during a sample run of the application. The AppInstallBlock contains an optional application-specific initialization code 3107. This code is needed when the default initialization procedure is insufficient to setup the local machine environment for that particular application.

[0453] The AppInstallBlock and the runtime data are packaged into the SAS by the Builder and then uploaded to the application server. After the SAS client is subscribed to an application and before the application is run for the first time, the AppInstallBlock is sent by the server to the client. The SAS client invokes the default initialization procedure and the optional application-specific initialization code 3107. Together, the default and the application-specific initialization procedure process the data in the AppInstallBlock to make the machine ready for streaming that particular application.

[0454] Data Type Definitions

[0455] The AppInstallBlock is divided into the following sections: header section 3101, variable section 3103, file section 3102, profile section 3105, prefetch section 3104, comment section 3106, and code section 3107. The header section 3101 contains general information about the AppInstallBlock. The information includes the total byte size and an index table containing size and offset into other sections. In a Windows version, the variable section consists of two registry tree structures to specify the registry entries added or removed from the OS environment. The file section 3102 is a tree structure consisting of the files copied to C drive during the application installation. The profile section 3105 contains the initial set of block reference sequences during Builder profiling of the application. The prefetch section 3104 consists of a subset of profiled blocks used by the Builder as a hint to the SAS client to prefetch initially. The comment section 3106 is used to inform the SAS client user of any relevant information about the application installation. Finally, the code section 3107 contains an optional program tailored for any application-specific installation not covered by the default streamed application installation procedure. In Windows version, the code section contains a Windows DLL. The following is a detailed description of each fields of the AppInstallBlock.

[0456] Note: Little endian format is used for all the fields spanning more than 1 byte. Also, BlockNumber specifies blocks of 4K byte size.

[0457] 1. Header Section

[0458] The header section 3103 contains the basic information about this AppInstallBlock. This includes the versioning information, application identification, and index into other sections of the file.

[0459] Core Header Structure

[0460] AibVersion [4 bytes]: Magic number or appInstallBlock version number (which identifies the version of the appInstallBlock structure rather than the contents).

- [0461] AppId [16 bytes]: this is an application identifier unique for each application. On Windows, this identifier is the GUID generated from the 'guidgen' program. AppId for Word on Win98 will be different from Word on WinNT if it turns out that Word binaries are different between NT and 98.
- [0462] VersionNo [4 bytes]: Version number. This allows us to inform the client that the appInstallBlock has changed for a particular appId. This is useful for changes to the AppInstallBlock due to minor patch upgrades in the application.
- [0463] ClientOSBitMap [4 bytes]: Client OS supported bitmap or ID: for Win2K, Win98, WinNT (and generally for other and multiple OSs).
- [0464] ClientOSServicePack [4 bytes]: For optional storage of the service pack level of the OS for which this appInstallBlock has been created. Note that when this field is set, the multiple OS bits in the above field ClientOSBitMap are not used.
- [0465] Flags [4 bytes]: Flags pertaining to AppInstallBlock
- [0466] Bit 0: Reboot—If set, the SAS client needs to reboot the machine after installing the AppInstallBlock on the client machine.
- [0467] Bit 1: Unicode—If set, the string characters are 2 bytes wide instead of 1 byte.
- [0468] HeaderSize [2 bytes]: Total size in bytes of the header section.
- [0469] Reserved [32 bytes]: Reserved spaces for future.
- [0470] NumberOfSections [1 byte]: Number of sections in the index table.
- [0471] This determines the number of entries in the index table structure described below:
- [0472] Index Table Structure: (Variable Number of Entries)
- [0473] SectionType [1 bytes]: The type of data described in section. 0=file section, 1=variable section, 2=prefetch section, 3=profile section, 4=comment section, 5=code section.
- [0474] SectionOffset [4 bytes]: The offset from the beginning of the file indicates the beginning of section.
- [0475] SectionSize [4 bytes]: The size in bytes of section.
- [0476] Variable Structure
- [0477] ApplicationNameIsAnsi [1 byte]: 1 if ansi, 0 if Unicode.
- [0478] ApplicationNameLength [4 bytes]: Byte size of the application name
- [0479] ApplicationName [X bytes]: Null terminating name of the application
- [0480] 2. File Section
- [0481] The file section 3102 contains a subset of the list of files needed by the application to run properly. This section does not enumerate files located in the standard application program directory. It consists of information about files copied into an 'unusual' directory during the installation of an application. If the file content is small (typically less than 1 MB), the file is copied to the client machine. Otherwise, the file is relocated to the standard program directory suitable for streaming. The file section data is a list of trees stored in a contiguous sequence of address spaces according to the pre-order traversal of the trees. A node in the tree can correspond to one or more levels of directories. A parent-child node pair is combined into a single node if the parent node has only a single child. Parsing the tree from the root of the tree to a leaf node results in a fully legal Windows pathname including the drive letter. Each entry of the node in the tree consists of the following structure:
- [0482] Directory Structure: (Variable Number of Entries)
- [0483] Flags [4 byte]: Bit 0 is set if this entry is a directory
- [0484] NumberOfChildren [2 bytes]: Number of nodes in this directory
- [0485] DirectoryNameLength [4 bytes]: Length of the directory name
- [0486] DirectoryName [X bytes]: Null terminating directory name
- [0487] Leaf Structure: (Variable Number of Entries)
- [0488] Flags [4 byte]: Bit 1 is set to 1 if this entry is a spoof or copied file name
- [0489] FileVersion [8 bytes]: Version of the file GetFileVersionInfo() if the file is win32 file image. Need variable file version size returned by GetFileVersionInfoSize(). Otherwise use file size or file modified time to compare which file is the later version.
- [0490] FileNameLength [4 bytes]: Byte size of the file name
- [0491] FileName [X bytes]: Null terminating file name
- [0492] DataLength [4 bytes]: Byte size of the data. If spoof file, then data is the string of the spoof directory. If copied file, then data is the content of the file
- [0493] Data [X bytes]: Either the spoof file name or the content of the copied file
- [0494] 3. Add Variable and Remove Variable Sections
- [0495] The add and remove variable sections 3103 contain the system variable changes needed to run the application. In a Windows system, each section consists of several number of registry subtrees. Each tree is stored in a contiguous sequence of address spaces according to the pre-order traversal of the tree. A node in the tree can correspond to one or more levels of directory in the registry. A parent-child node pair is combined into a single node if the parent node has only a single child. Parsing the tree from the root of the tree to a leaf node results in a fully legal key name. The order of the trees is shown here.

[0496] a. Registry Subsection:

- [0497]** 1. "HKCR": HKEY_CLASSES_ROOT
- [0498]** 2. "HKCU": HKEY_CURRENT_USER
- [0499]** 3. "HKLM": HKEY_LOCAL_MACHINE
- [0500]** 4. "HKUS": HKEY_USERS
- [0501]** 5. "HKCC": HKEY_CURRENT_CONFIG

[0502] Tree Structure: (5 entries)

- [0503]** ExistFlag [1 byte]: Set to 1 if this tree exist, 0 otherwise.
- [0504]** Key or Value Structure entries [X bytes]: Serialization of the tree into variable number key or value structures described below.

[0505] Key Structure: (Variable Number of Entries)

- [0506]** KeyFlag [1 byte]: Set to 1 if this entry is a key or 0 if it's a value structure
- [0507]** NumberOfSubchild [4 bytes]: Number of subkeys and values in this key directory
- [0508]** KeyNameLength [4 bytes]: Byte size of the key name
- [0509]** KeyName [X bytes]: Null terminating key name

[0510] Value Structure: (Variable Number of Entries)

- [0511]** KeyFlag [1 byte]: Set to 1 if this entry is a key or 0 if it's a value structure
- [0512]** ValueType [4 byte]: Type of values from the Win32 API function RegQueryValueEx(): REG_SZ, REG_BINARY, REG_DWORD, REG_LINK, REG_NONE, etc . . .
- [0513]** ValueNameLength [4 bytes]: Byte size of the value name
- [0514]** ValueName [X bytes]: Null terminating value name
- [0515]** ValueDataLength [4 bytes]: Byte size of the value data
- [0516]** ValueData [X bytes]: Value of the Data

[0517] In addition to registry changes, an installation in a Windows system may involve changes to the ini files. The following structure is used to communicate the ini file changes needed to be done on the SAS client machine. The ini entries are appended to the end of the variable section after the five registry trees are enumerated.

[0518] b. INI Subsection

- [0519]** NumFiles [4 bytes]: Number of INI files modified.

[0520] File Structure: (Variable Number of Entries)

- [0521]** FileNameLength [4 bytes]: Byte length of the file name
- [0522]** FileName [X bytes]: Name of the INI file
- [0523]** NumSection [4 bytes]: Number of sections with the changes

[0524] Section Structure: (Variable Number of Entries)

- [0525]** SectionNameLength [4 bytes]: Byte length of the section name
- [0526]** SectionName [X bytes]: Section name of an INI file
- [0527]** NumValues [4 bytes]: Number of values in this section
- [0528]** Value Structure: (Variable Number of Entries)
 - [0529]** ValueLength [4 bytes]: Byte length of the value data
 - [0530]** ValueData [X bytes]: Content of the value data

[0531] 4. Prefetch Section

[0532] The prefetch section 3104 contains a list of file blocks. The Builder profiler determines the set of file blocks critical for the initial run of the application. This data includes the code to start and terminate the application. It includes the file blocks containing frequently used commands. For example, opening and saving of documents are frequently used commands and should be prefetched if possible. Another type of block to include in the prefetch section are the blocks associated with frequently accessed directories and file metadata in this directory. The prefetch section is divided into two subsections. One part contains the critical blocks that are used during startup of the streamed application. The second part consists of the blocks accessed for common user operations like opening and saving of document. The format of the data is described below:

[0533] a. Critical Block Subsection:

- [0534]** NumCriticalBlocks [4 bytes]: Number of critical blocks.
- [0535]** Block Structure: (Variable Number of Entries)
 - [0536]** FileNumber [4 bytes]: File Number of the file containing the block to prefetch
 - [0537]** BlockNumber [4 bytes]: Block Number of the file block to prefetch

[0538] b. Common Block Subsection:

- [0539]** NumCommonBlocks [4 bytes]: Number of critical blocks.

[0540] Block Structure: (Variable Number of Entries)

- [0541]** FileNumber [4 bytes]: File Number of the file containing the block to prefetch
- [0542]** BlockNumber [4 bytes]: Block Number of the file block to prefetch

[0543] 5. Profile Section

[0544] The profile section 3105 consists of a reference sequence of file blocks accessed by the application at runtime. Conceptually, the profile data is a two dimensional matrix. Each entry [row, column] of the matrix is the frequency, a block row is followed by a block column. In any realistic applications of fair size, this matrix is very large and sparse. The proper data structure must be selected to

store this sparse matrix efficiently in required storage space and minimize the overhead in accessing this data structure access.

[0545] The section is constructed from two basic structures: row and column structures. Each row structure is followed by N column structures specified in the Number-Columns field. Note that this is an optional section. But with appropriate profile data, the SAS client prefetcher performance can be increased.

[0546] Row Structure: (Variable Number of Entries)

[0547] FileNumber [4 bytes]: File Number of the row block

[0548] BlockNumber [4 bytes]: Block Number of the row block

[0549] NumberColumns [4 bytes]: number of blocks that follows this block. This field determines the number of column structures following this field.

[0550] Column Structure: (Variable Number of Entries)

[0551] FileNumber [4 bytes]: File Number of the column block

[0552] BlockNumber [4 bytes]: Block Number of the column block

[0553] Frequency [4 bytes]: frequency the row block is followed by column block

[0554] 6. Comment Section

[0555] The comment section 3106 is used by the Builder to describe this ApplInstallBlock in more detail.

[0556] CommentLengthIsAnsi [1 byte]: 1 if string is ansi, 0 if Unicode format.

[0557] CommentLength [4 bytes]: Byte size of the comment string

[0558] Comment [X bytes]: Null terminating comment string

[0559] 7. Code Section

[0560] The code section 3107 consists of the application-specific initialization code needed to run on the SAS client to setup the client machine for this particular application. This section may be empty if the default initialization procedure in the SAS client is able to setup the client machine without requiring any application-specific instructions. On the Windows system, the code is a DLL file containing two exported function calls: Install(), Uninstall(). The SAS client loads the DLL and invokes the appropriate function calls.

[0561] CodeLength [4 bytes]: Byte size of the code

[0562] Code [X bytes]: Binary file containing the application-specific initialization code. On Windows, this is just a DLL file.

[0563] 8. LicenseAgreement Section

[0564] The Builder creates the license agreement section 3108. The SAS client displays the license agreement text to the end-user before the application is started for the first time. The end-user must agree to all licensing agreement set by the software vendor in order to use the application.

[0565] LicenseTextIsAnsi [1 byte]: 1 if ansi, 0 if Unicode format.

[0566] LicenseTextLength [4 bytes]: Byte size of the license text

[0567] LicenseAgreement [X bytes]: Null terminating license agreement string

Client Installation and Execution of Streamed Applications

[0568] Summary

[0569] This section describes the process of installing and uninstalling streamed application on the client machine. With respect to FIG. 32, the lifecycle of the Application Install Block is shown. The Application Stream Builder 3202 takes original application files 3201 and produces a corresponding Application Install Block and Stream Application Set 3203. These two files get installed onto the application servers 3206. On the right side of the drawing, it shows how either the administrator or the user can subscribe to the application from either the client computer 3208 or an administration computer 3207. Once the user logons onto the client computer 3208, the license and the AIB 3203 are acquired from the license 3205 and application servers 3206, respectively.

[0570] The following are features of a preferred embodiment of the invention:

[0571] 1. The streamed application installation process installs just the description of the application, not the total content of the application. After installing such description, the client system looks and feels similar to having installed the same app using a non-streamed method. This has the following benefits:

[0572] a. Takes a very small fraction of the time to install the application.

[0573] b. Takes a very small fraction of the disk space.

[0574] c. Client does not have to wait for the entire application to be downloaded. This is particularly important to users with slow network connections.

[0575] The application description is subsequently uninstalled without requiring deleting the total contents of the application. This has the benefit that it takes a very small fraction of the time to uninstall the application.

[0576] 2. Enhancing streamed application's performance by:

[0577] a. Copying small portions of application's code and data (pages) that are critical to performance.

[0578] b. Providing client with the initial profile data that can be used to perform pre-fetching.

[0579] This has the following benefits:

[0580] 1. User experiences smooth and predictable application launch.

[0581] 2. Scalability of Application servers increases by reducing the number of client connections.

[0582] 3. An administrator can arrange applications to be installed automatically on client computers. Administrator can also arrange the installation on various client computers simultaneously without being physically present on each client computer. This has the following benefits:

[0583] a. Users are not burdened with the process of installing streamed applications.

[0584] b. Reduced administration expense.

[0585] Overview of Components Relevant to the Install Process

[0586] Subscription Server 3204: allows users to create accounts & to rent.

[0587] License Server 3205: authenticates users & determines licensing rights to applications.

[0588] Application Server 3206: provides application bits to licensed users securely & efficiently.

[0589] Application Install Manager—a component installed on the streaming client that is responsible for installing and uninstalling streamed applications.

[0590] Application Install Block (AIB) 3203—a representation of what gets installed on the client machine when a streamed application is installed. It contains portions of the application that are responsible for registering the application with the client operating system and other data that enhances the execution of streamed application.

[0591] Application Stream Builder 3202—preprocesses apps & prepares files to be installed on Application Server and data, such as AIB, to be installed by Client Application Installer.

[0592] Stream Application Set 3203—a method of representing the total content of the application in a format that is optimal for streaming.

[0593] Client Streaming File System—integrates client exec with paging from a special file system backed by remote network-accessed server-based store

[0594] Application Install Block (AIB)

[0595] Installing and un-installing a stream application requires an understanding of what AIB is and how it gets manipulated by the various components in the overall streaming system. AIB is physically represented as a data file with various different sections. Its contents include:

[0596] Streamed application name and identification number.

[0597] Software License Agreement.

[0598] Registry spoof set.

[0599] File spoof set.

[0600] Small number of application pages—initial cache contents.

[0601] Application Profile Data.

[0602] AIB Lifecycle

[0603] The following describes the AIB lifecycle:

[0604] 1. Using the process described in the section above concerning converting apps for stream delivery and subsequent execution, an application install

block is created by the Application Stream Builder. Initially, there will be one AIB per application, however, as the application evolves via patches and service packs, new AIBs may need to be generated.

[0605] 2. Using a process described in the section above regarding server-side performance optimization, AIB will get hosted by the application servers.

[0606] 3. “Subscribing” the application by communicating with the subscription server. Subscribing to an application requires a valid account with the ASP. Either the user or an administrator acting on the user’s behalf can subscribe the application. In addition, the application can be subscribed to from any computer on the Internet, not just the client machine where the application will be eventually installed. This allows an administrator to subscribe applications for a group of users without worrying about individual client machines.

[0607] 4. The client machine acquires the license for the application from the license server. If the application was subscribed from the client machine itself, this step will happen automatically after subscribing to the application. If the subscription happened from a different machine, e.g., the administrator’s machine, this step will happen when the user logs on the client machine. As an acknowledgment of having a valid license, the license server gives the client an encrypted access token.

[0608] 5. Fetch the contents of AIB from the application server. This step is transparent and happens immediately after the preceding step. Since application server requires the client to possess a valid access token, it ensures that only subscribed and licensed users can install the streamed application.

[0609] 6. The Application Install Manager (AIM) performs the act of installing the application information, as specified by the AIB, on the client system.

[0610] Installing a Streamed Application

[0611] AIM downloads AIB from the application server and takes the necessary steps in installing the application description on the client system. It extracts pieces of information from AIB and sends messages to various other components (described later) to perform the installation. AIM also creates an Install-Log that can be used when un-installing the streamed application.

[0612] 1. Display a license agreement to the user and wait for the user to agree to it.

[0613] 2. Extract File Spoof Data and communicate that to the Client File Spoofer. The list of files being spoofed will be recorded in the Install-Log.

[0614] 3. Extract Registry Spoof Data and communicate that to the Client Registry Spoofer. The list of Registries being spoofed will be recorded in the Install-Log.

[0615] 4. Extract Initial Cache Content and communicate that to the Client Prefetch Unit.

[0616] 5. Extract Profile Data and communicate that to the Client Prefetch Unit.

[0617] 6. Save the Install-Log in persistent storage.

[0618] Un-Installing a Streamed Application

[0619] Un-installation process relies on the Install-Log to know what specific items to un-install. Following steps are performed when un-installing and application:

[0620] 1. Communicate with the Client Registry Spoofer to remove all registries being spoofed for the application being un-installed.

[0621] 2. Communicate with the Client File Spoofer to disable all files being spoofed for the application being un-installed.

[0622] 3. Communicate with the Client Prefetch Unit to remove all Profile Data for the application being un-installed.

[0623] 4. Communicate with the Client Cache Manager to remove all pages being cached for the application being un-installed.

[0624] 5. Delete the Install-Log.

[0625] Client File Spoofer

[0626] A file spoofer component is installed on the client machine and is responsible for redirecting file accesses from a local file system to the streaming file system. The spoofer operates on a file spoof database that is stored persistently on the client system; it contains a number of file maps with following format:

[0627] [Original path of a local file] ↔ [New path of a file on streaming drive]

[0628] Where "↔" indicates a bi-directional mapping between the two sides of the relationship shown.

[0629] When a streamed application is installed, the list of new files to spoof (found in AIB) is added to the file spoof database. Similarly, when a streamed application is un-installed, a list of files being spoofed for that application is removed from the file spoof database.

[0630] On clients running the Windows 2000 Operating System, the file spoofer is a kernel-mode driver and the spoof database is stored in the registry.

[0631] Client Registry Spoofer

[0632] The Registry Spoofer intercepts all registry calls being made on the client system and re-directs calls manipulating certain registries to an alternate path. Effectively, it is mapping the original registry to an alternate registry transparently. Similar to the client file spoofer, the registry spoofer operates on a registry spoof database consisting entries old/new registry paths. The database must be stored in persistent storage.

[0633] When a streamed application is installed, the list of new registries to spoof (found in AIB) is added to the registry spoof database. Upon un-installation of a streamed application, its list of spoofed registries is removed from the registry spoof database.

[0634] On clients running the Windows 2000 Operating System, the registry spoofer is a kernel-mode driver and the registry spoof database is stored in the registry.

[0635] Client Prefetch Unit

[0636] In a streaming system, it is often a problem that the initial invocation of the application takes a lot of time because the necessary application pages are not present on the client system when needed. A key aspect of the client

install is that by using a client prefetch unit, a system in accordance with the present invention significantly reduces the performance hit associated with fetching. The Client Prefetch Unit performs two main tasks:

[0637] 1. Populate Initial Cache Content,

[0638] 2. Prefetch Application Pages.

[0639] Initial Cache Content

[0640] The Application Stream Builder determines the set of pages critical for the initial invocation and packages them as part of the AIB. These pages, also known as initial cache content, include:

[0641] Pages required to start and stop the application,

[0642] Contents of frequently accessed directories,

[0643] Application pages performing some of the most common operations within application. For example, if Microsoft Word is being streamed, these operations include: opening & saving document files & running a spell checker.

[0644] When the Stream Application is installed on the client, these pages are put into the client cache; later, when the streamed application is invoked, these pages will be present locally and network latency is avoided.

[0645] In preparing the Prefetch data, it is critical to manage the trade off of how many pages to put into AIB and what potential benefits it brings to the initial application launch. The more pages that are put into prefetch data, the smoother the initial application launch will be; however, since the AIB will get bigger (as a result of packing more pages in it), users will have to wait longer when installing the streamed application. In a preferred embodiment of the invention, the size of the AIB is limited to approximately 250 KB.

[0646] In an alternative embodiment of the invention the AIB initially includes only the page/file numbers and not the pages themselves. The client then goes through the page/file numbers and does paging requests to fetch the indicated pages from the server.

[0647] Prefetch Application Pages

[0648] When the streaming application executes, it will generate paging requests for pages that are not present in the client cache. The client cache manager must contact the application server and request the page in question. The invention takes advantage of this opportunity to also request additional pages that the application may need in the future. This not only reduces the number of connections to the application server, and overhead related to that, but also hides the latency of cache misses.

[0649] The application installation process plays a role in the pre-fetching by communicating the profile data present in the AIB to the Client Prefetch Unit when the application is installed. Upon un-installation, profile data for the particular application will be removed.

Caching of Streamed Application Pages Within the Network

[0650] Summary

[0651] This section describes how collaborative caching is employed to substantially improve the performance of a client server system in accordance with the other aspects of the present invention. Specifically, particular caching configurations and an intelligent way to combine these caching configurations are detailed.

[0652] Collaborative Caching Features:

[0653] Using another client's cache to get required pages/packets (Peer Caching)

[0654] Using an intermediate proxy or node to get required pages/packets (Proxy Caching)

[0655] Using a broadcasting or multicasting mechanism to make a request (Multicast)

[0656] Using a packet based protocol to send requested pages/packets rather than a stream based one. (Packet Protocol)

[0657] Using concurrency to request a page through all three means (Peer Caching or Proxy Caching or the actual server) to improve performance (Concurrent Requesting).

[0658] Using heuristical algorithms to use all three ways to get the required pages (Smart Requesting).

[0659] These features have the following advantages:

[0660] These ideas potentially improve the performance of the client, i.e., they reduce the time a client takes to download a page (Client Performance).

[0661] These ideas improve the scalability of the server because the server gets fewer requests, i.e., requests which are fulfilled by a peer or a proxy don't get sent to the server. (Server Scalability)

[0662] These allow a local caching mechanism without needing any kind of modification of local proxy nodes or routers or even the servers. The peer-to-peer caching is achieved solely through the co-operation of two clients. (Client Only Implementation)

[0663] These ideas allow a client to potentially operate "offline" i.e., when it is not getting any responses from the server (Offline Client Operation).

[0664] These ideas allow the existing network bandwidth to be used more effectively and potentially reduce the dependency of applications on higher bandwidth (Optimal Use of Bandwidth).

[0665] These ideas when used in an appropriate configuration allow each client to require a smaller local cache but without substantially sacrificing the performance that you get by local caching. An example is when each client "specializes" in caching pages of a certain kind, e.g., a certain application. (Smaller Local Cache).

[0666] These ideas involve new interrelationships—peer-to-peer communication for cache accesses; or new configurations—collaborative caching. The reason this is called collaborative is because a group of clients can collaborate in caching pages that each of them needs.

[0667] Aspects of Collaborative Caching

[0668] 1. Peer Caching: A client X getting its pages from another client Y's local cache rather than its (X's) own or from the server seems to be a new idea. Major advantages: client performance, server scalability, client only implementation, offline client operation, optimal use of bandwidth, smaller local cache.

[0669] 2. Proxy Caching: The client getting its pages from an intermediate proxy which either serves the page from the local cache or from another intermediate proxy or the remote server (if none of the intermediate proxies has the page) is unique, at a minimum, for the pages of a streamed application. Major advantages: client performance, server scalability, offline client operation (to some extent), optimal use of bandwidth, smaller local cache.

[0670] 3. Multicast: Using multicasting (or selective broadcasting) considerably reduces peer-to-peer communication. For every cache request there is only one packet on the network and for every cache response there is potentially only one packet on the network in some configurations. This definitely helps reduce network congestion. Major advantages: client performance, server scalability, client only implementation, offline client operation, optimal use of bandwidth

[0671] 4. Packet Protocol: Because only datagram packets are used to request or respond to cache pages this saves the overhead of opening stream-based connections such as a TCP connection or an HTTP connection. Major advantages: client performance, client only implementation, offline client operation, and optimal use of bandwidth.

[0672] 5. Concurrent Requesting: If concurrent or intelligently staggered requests through all three means are issued to request a single page, the client will be able to receive the page through the fastest means possible for that particular situation. Major advantages: client performance, server scalability, offline client operation, and optimal use of bandwidth

[0673] 6. Smart Requesting: An adaptive or "smart" algorithm can be used to further enhance the overall performance of the client-server system. In this algorithm, the client uses the data of how past requests were processed to "tune" new requests. For example, if the client's past requests were predominantly served by another client, i.e., Peer Caching worked, then for new page requests the client would first try to use Peer Caching, and wait some time before resorting to either Proxy Caching or direct server request. This wait time can again be calculated in an adaptive fashion. Major advantages: client performance, server scalability, client only implementation, offline client operation, and optimal use of bandwidth.

[0674] The concepts illustrated herein can be applied to many different problem areas. In all client-server implementations where a server is serving requests for static data, e.g., code pages of a streamed application or static HTML pages

from a Website, the approaches taught herein can be applied to improve the overall client-server performance. Even if some of the protocols or configurations described in this document are not supported by the underlying network, it does not preclude the application of other ideas described herein that do not depend on such features. For example, if multicast (or selective broadcast) is not supported, ideas such as Concurrent Requesting or Smart Requesting can still be used with respect to multiple servers instead of the combination of a server, peer, and proxy. Also the use of words like Multicast does not restrict the application of these ideas to multicast based protocols. These ideas can be used in all those cases where a multicast like mechanism, i.e., selective broadcasting is available. Also note that the description of these ideas in the context of LAN or intranet environment does not restrict their application to such environments. The ideas described here are applicable to any environment where peers and proxies, because of their network proximity, offer significant performance advantages by using Peer Caching or Proxy Caching over a simple client-server network communication. In that respect, the term LAN or local area network should be understood to mean more generally as a collection of nodes that can communicate with each other faster than with a node outside of that collection. No geographical or physical locality is implied in the use of the term local area network or LAN.

[0675] Peer Caching

[0676] Referring to FIG. 33, how multiple peers collaborate in caching pages that are required by some or all of them is shown.

[0677] The main elements shown are:

[0678] Client 13301 through Client 63306 in an Ethernet LAN 3310.

[0679] Router 1 and the local proxy serving as the Internet gateway 3307. Note that it does not matter whether Router 1 and the proxy are one computer or two different ones.

[0680] Other routers from router 2 through router N 3308 that are needed to connect the LAN 3310 to the Internet 3311.

[0681] A remote server 3309 (that is reachable only by going over the Internet 3311) that is serving the pages that the above mentioned clients need.

[0682] A cloud that symbolizes the complexity of the Internet 3311 and potentially long paths taken by packets.

[0683] Client 23302 needs a page that it does not find in its local cache. It then decides to use the mechanism of Peer Caching before attempting to get the page from the local proxy (or the actual server through the proxy). The actual sequence of events is as follows:

[0684] 1. Client 23302 sends a request for the page it needs. This request is sent as a multicast packet to a predetermined multicast address and port combination. Lets call this multicast address and port combination as M.

[0685] 2. The multicast packet is received by all the clients that have joined the group M. In this case all six clients have joined the group M.

[0686] 3. Client 53305 receives the request and it records the sender's, i.e., Client 2's 3302, address and port combination. Let's assume this address and port combination is A. Client 53305 processes the request and looks up the requested page in its own cache. It finds the page.

[0687] 4. Client 53305 sends the page to address A (which belongs to Client 23302) as a packet.

[0688] 5. Client 23302 receives the page it needs and hence does not need to request the server for the page.

[0689] Proxy Caching

[0690] With respect to FIG. 43, a transparent proxy and how clients use it to get pages is shown. Again the elements here are the same as in the previous figure:

[0691] Client 13401 through Client 63406 in an Ethernet LAN 3410.

[0692] Router 1 and the local proxy serving as the Internet gateway 3407. Note that it does not matter whether Router 1 and the proxy are one computer or two different ones.

[0693] Other routers from router 2 through router N 3408 that are needed to connect the LAN 3410 to the Internet 3411.

[0694] A remote server 3409 (that is reachable only by going over the Internet 3411) that is serving the pages that the above mentioned clients need.

[0695] A cloud that symbolizes the complexity of the Internet 3411 and potentially long paths taken by packets.

[0696] Assume Peer Caching is either not enabled or did not work for this case. When Client 23402 needs a page, it makes a request to the proxy 3407. The proxy 3407 finds the page in its local cache and returns it to Client 23402. Because of this, the request did not go to the remote server 3409 over the Internet 3411.

[0697] Multicast and Packet Protocol Within a LAN

[0698] Referring to FIG. 35, the role played by multicast and unicast packets in Peer Caching is shown. The example of the drawing "Peer Caching" is used to explain multicast. Here Client 23502 has the IP address 10.0.0.2 and it opens port 3002 for sending and receiving packets. When Client 23502 needs a page and wants to use Peer Caching to get it, it forms a request and sends it to the multicast address and port 239.0.0.1:2001. All the other clients in the LAN 3508 that support Peer Caching have already joined the group 239.0.0.1:2001 so they all receive this packet.

[0699] Client 53505 receives this packet and it records the sender address (10.0.0.2:3002 in this case). It looks up the requested page and finds it in its local cache. It sends the page as a response packet to the address 10.0.0.2:3002.

[0700] Client 23502 receives this response packet since it was waiting at this port after sending the original multicast request. After ensuring the validity of the response, it retrieves the page it needs.

[0701] Note that more than one client can respond to the original multicast request. However Client 23502 can discard all the later responses, since it has already received the page it needed.

[0702] Concurrent Requesting—Proxy First

[0703] With respect to FIG. 36, one particular case of how Concurrent Requesting is used is shown. This is a timeline of events that take place in the client. When a client first needs a page, it does not know whether it is going to get any responses through Peer Caching or not. Hence it issues a request to the proxy (or the server through the proxy) as soon as it needs the page. Then it issues a request using the Peer Caching mechanism. If there is indeed a peer that can return the page requested, the peer presumably could return the page faster than the proxy or the server. If this happens, the client may decide to use Peer Caching mechanism before attempting to get the page from the proxy or the server. The timeline essentially describes the following sequence of events:

[0704] 1. At time $t=0$, a page p is needed by the client 3601.

[0705] 2. The client looks up its local cache, and it doesn't find page p .

[0706] 3. At time $t=T1$, it decides to send a request to the proxy to get the page 3602.

[0707] 4. After a delay of amount D_p 3603, at time $t=T2$ it also sends a request for the page p through the mechanism of Peer Caching 3604. Note that D_p 3603 can be zero, in which case $T1=T2$.

[0708] 5. At time $t=T3$, a response is received from another peer that contains the page p that this client needs 3606. Thus the response time of the Peer Caching mechanism is $R_p=T3-T2$ 3605.

[0709] 6. At time $t=T4$, a response from the proxy/server is received that contains the page p 3608. Hence the response time of the proxy/server is $R_s=T4-T1$ 3607.

[0710] Note that since, $R_p < R_s$, the client will increase the weighting for Peer Caching in all of its future queries. That means it will decrease D_p , and if D_p is already zero, it will increase D_s (the delay before requesting proxy/server). On the other hand, if $R_p > R_s$ or if R_p were infinity, it will increase its weighting for proxy/server requesting. This is part of Smart Requesting that is explained elsewhere in this document.

[0711] Concurrent Requesting—Peer Caching First

[0712] Referring to FIG. 37, in contrast to the previous figure, the client has decided to use Peer Caching before requesting the proxy. So the sequence of events is as follows:

[0713] 1. At time $t=0$, a page p is needed by the client 3701.

[0714] 2. The client looks up its local cache, and it doesn't find page p .

[0715] 3. At time $t=T5$, it decides to send a request for the page p through the mechanism of Peer Caching 3702.

[0716] 4. After a delay of amount D_s 3703, at time $t=T6$ it also sends a request for the page p to the proxy/server. Note that D_s can be zero, in which case $T5=T6$.

[0717] 5. At time $t=T7$, a response is received from another peer that contains the page p that this client needs 3706. Thus the response time of the Peer Caching mechanism is $R_p=T7-T5$ 3705.

[0718] 6. At time $t=T8$, a response from the proxy/server is received that contains the page p 3708. Hence the response time of the proxy/server is $R_s=T8-T6$ 3707.

[0719] As described in the previous drawing, the client increases the weighting of Peer Caching even more because it got a response through Peer Caching long before it got a response from the proxy/server. As a result of the increases weighting the delay D_s is increased even more.

[0720] Concurrent Requesting—Peer Caching Only

[0721] With respect to FIG. 38, in contrast with FIG. 37, the client has increased D_s 3805 (the delay before requesting a proxy/server) so much, that if a page is received before the expiry of the delay D_s 3805, the client does not even make a request to the proxy/server. The shaded area 3806 shows the events that do not take place because of this.

[0722] Client-Server System with Peer and Proxy Caching

[0723] Referring to FIG. 39, a system level drawing that gives a system context for all the other figures and discussion in this document is shown. This drawing illustrates all three ways in which a client gets its page requests fulfilled. Note that:

[0724] Client 23902 gets its page request fulfilled through Peer Caching, i.e., multicast request.

[0725] Client 13901 gets its page request fulfilled through Proxy Caching, i.e., the proxy 3907 finds the page in its cache and returns it.

[0726] Client 33903 has to go to the server 3909 over the Internet 3908 to get its page request fulfilled.

[0727] Collaborative Caching Details

[0728] In a typical client-server model, caching could be used to improve the performance of clients and scalability of servers. This caching could be:

[0729] Local to the client where the client itself locally stores the pages it had received from the server in the past. Then the client would not need to request the proxy/server for any page that resides in the local cache as long as the locally cached page is "valid" from the server point of view.

[0730] On a proxy node that can be any node along the path taken by a packet that goes from the client to the server. The closer this proxy node is to the client the more improvement in the performance you get.

[0731] On a peer node, that is on another client. In this case, the two clients (the requesting client as well as the serving client) are on the same LAN or intranet, so that the travel time of a packet between

the two nodes is considerably smaller as compared to the travel time of the packet from one of the clients to the server.

[0732] As far as caching is concerned, this section details the new ideas of Peer Caching and Proxy Caching. In addition, it also details the new ideas of Concurrent Requesting and Smart Requesting. The preferred approaches for implementing these ideas are also described here and these are Multicast and Packet Protocol.

[0733] The idea of Peer Caching is nothing but a client X taking advantage of the fact that a peer, e.g., say another client Y, on its LAN had, in the past, requested a page that X is going to request from its server. If the peer Y has that page cached locally on its machine, then X could theoretically get it much faster from Y than getting it from the server itself. If an efficient mechanism is provided for the two clients X and Y to collaborate on this kind of cache access, then that will offer many advantages such as: Client Performance, Server Scalability, Client Only Implementation, Offline Client Operation, Optimal Use of Bandwidth, Smaller Local Cache. Note that two clients were considered only as an example, the idea of Peer Caching is applicable to any number of peers on a LAN.

[0734] The idea of Multicast is to use the multicast protocol in the client making a Peer Caching request. Multicast can be briefly described as "selective broadcasting"—similar to radio. A radio transmitter transmits "information" on a chosen frequency, and any receiver (reachable by the transmitter, of course) can receive that information by tuning to that frequency. In the realm of multicast, the equivalent of a radio frequency is a multicast or class D IP address and port. Any node on the net can send datagram packets to a multicast IP address+port. Another node on the net can "join" that IP address+port (which is analogous to tuning to a radio frequency), and receive those packets. That node can also "leave" the IP address+port and thereby stop receiving multicast packets on that IP address+port.

[0735] Note that multicast is based on IP (Internet Protocol) and is vendor neutral. Also, it is typically available on the Ethernet LAN and, if routers supported it, it can also go beyond the LAN. If all the routers involved in a node's connection to the Internet backbone supported multicast routing, multicast packets theoretically could go to the whole Internet except the parts of the Internet that do not support multicast routing.

[0736] The use of multicast allows a client to not have to maintain a directory of peers that can serve its page requests. Also because of multicast there is only one packet per page request. Any peer that receives the request could potentially serve that request, so by using a multicast based request there are multiple potential servers created for a page request but only one physical packet on the network. This contributes substantially in reducing network bandwidth, but at the same time increasing peer accessibility to all the peers. When implemented properly, the packet traffic due to Peer Caching will be proportional to the number of clients on the network participating in Peer Caching.

[0737] An idea related to Multicast is Packet Protocol. Note that Multicast itself is a packet-based protocol as opposed to connection based. The idea of Peer Caching here is described using Multicast and Packet Protocol. The Peer

Caching request is sent as a multicast request and the response from a peer to such a request is also sent as a packet (not necessarily a multicast packet). Sending packets is much faster than sending data through a connection-based protocol such as TCP/IP, although using packet-based protocol is not as reliable as using connection-based one. The lack of reliability in Packet Protocol is acceptable since Peer Caching is used only to improve overall performance of the Client-Server system rather than as a primary mechanism for a client to get its pages. The underlying assumption made here is that a client could always get its pages from the server, if Peer Caching or Proxy Caching does not work for any reason.

[0738] The ideas of Concurrent Requesting and Smart Requesting describe how Peer Caching, Proxy Caching and client-server access could be combined in an intelligent fashion to achieve optimal performance of the whole Client-Server system. As part of Concurrent Requesting, a client is always prepared to make concurrent requests to get the page it needs in the fastest way possible. Concurrent Requesting would require the use of objects such as threads or processes that would allow one to programmatically implement Concurrent Programming. This document assumes the use of threads to describe a possible and preferred way to implement Concurrent Requesting.

[0739] The idea of Smart Requesting includes using an adaptive algorithm to intelligently stagger or schedule requests so that a client, even while using Concurrent Requesting, would not unnecessarily attempt to get a page through more than one means. An example of this is when a client has consistently gotten its page requests fulfilled through Peer Caching in the past. It would come to depend on Peer Caching for future page requests more than the other possible means. On the other hand, if Peer Caching has not worked for that client for some time, it would schedule a proxy request before a Peer Caching request. Smart Requesting involves dynamically calculating the delays D_p and D_s based how well Peer Caching and Proxy Caching has worked for the client. Please see FIGS. 36 through 38.

[0740] The following is an algorithmic description using pseudo-code of an illustrative embodiment.

[0741] startOurClient is a function that is invoked initially when the client is started.

```
void startOurClient() {
    Initialize the global variable delay to appropriate value based on a
    predefined policy. When delay is positive, it signifies the amount of
    time to wait after Proxy Caching before Peer Caching is attempted;
    and when delay is negative it signifies the amount of time to wait
    after Peer Caching before Proxy Caching is attempted. As an
    example:
    delay = 50;
    Start a thread for peer responses (i.e., Peer Caching server) with
    thread function as peerServer;
}
getPage function
```

[0742] The function getPage is called by the client's application to get a page. This function looks up the local cache and if the page is not found, attempts to get the page from a peer or proxy/server using the ideas of Concurrent Requesting and Smart Requesting.

```

void getPage(PageldType pageld) {
    if pageld present in the local cache then {
        retrieve it and return it to the caller;
    }
    if (delay > 0) {
        myDelay = delay;
        Call requestProxy(pageld);
    }
    else {
        myDelay = -delay;
        Call requestPeer(pageld);
    }
    Wait for gotPage event to be signaled for a maximum of myDelay
    milliseconds;
    If the page was obtained as indicated by gotPage being signaled {
        Modify delay appropriately i.e., if the page was obtained through
        Proxy Caching increment delay else decrement it;
        Return the page;
    }
    if (delay > 0) {
        Call requestPeer(pageld);
    }
    else {
        Call requestProxy(pageld);
    }
    Wait for the page to come through either methods;
    Depending on how the page came (through Proxy Caching or Peer
    Caching) increment or decrement delay;
    Return the page;
}
requestProxy function

```

[0743] The function requestProxy sends a page request to the proxy and starts a thread that waits for the page response (or times out). The function proxyResponse is the thread function that waits for the response based on the arguments passed to it.

```

void requestProxy(pageld) {
    Send a page request for pageld to a predefined proxy/server as per the
    proxy/server protocol;
    Start a thread with the thread function proxyResponse that waits
    for the response to the request - the function proxyResponse is
    passed arguments: the socket X where it should wait and pageld.
}

void proxyResponse(socket X, pageld) {
    Wait at the socket X for a response with a timeout of time TY;
    If a response was received at socket X {
        Uncompress the packet if necessary;
        Validate the packet and ensure that this is a
        valid response to the request and has the page requested (i.e.,
        match the pageld);
    }
    else {
        // this is time out: didn't receive any
        // response in time TY
        Set appropriate indicator to indicate time-out;
    }
    Signal an event to signify completion of this thread;
}
requestPeer and peerResponse functions

```

[0744] The function requestPeer is similar to requestProxy except that it sends a page request to peers and starts a thread that waits for the page response (or times out). The function peerResponse is the thread function that waits for the response based on the arguments passed to it.

```

Void requestPeer(pageld) {
    Create a UDP socket X bound to port 3002;
    Compose a packet that consists of:
        * a code indicating that this is a request for a page
        * Some kind of an identifier that uniquely identifies the page
        wanted such as the URL
        * other info such as security information or access validators
    Send this packet as a multicast packet to 239.0.0.1:2001 through
    the socket X created above;
    Create a thread with the thread function peerResponse and pass
    socket X and pageld as arguments to it;
}

Void peerResponse(socket X, pageld) {
    Wait at the socket X for a response with a timeout of time TX;
    If a packet was received at socket X {
        Uncompress the packet if necessary;
        Validate the packet and ensure that this is a
        valid response to the request and has the page requested (i.e.,
        match the pageld);
    }
    else {
        // this is time out: didn't receive any
        // response in time TX
        Set appropriate indicator to indicate time-out;
    }
    Signal an event to signify completion of this thread;
}
peerServer function

```

[0745] The function peerServer described below serves page requests received through Peer Caching as multicast packets. The function below describes how this thread would work:

```

void peerServer() {
    Create a multicast socket M bound to port 2001;
    Have M "join" the IP address 239.0.0.1;
    while (not asked to terminate) {
        Wait at M for a multicast packet;
        If a packet is received then {
            Store the source IP addr in S along with the source port number in B;
            Validate the packet that it is a valid request for a page that can be
            served (with valid security credentials);
            Look up the page id in the local client cache;
            If the page is found {
                Compose a packet that contains the pageld of the
                page as well as the page contents to send;
                Optionally compress the packet before sending;
                Send this packet to the IP address S at port B;
            }
        }
    }
}

```

Piracy Prevention for Streamed Applications

[0746] Summary

[0747] The details presented in this section describe new techniques of the invention that have been developed to combat software piracy of applications provided over networks, in situations where an ASP's clients' machines execute the software applications locally. The remote ASP server must make all the files that constitute an application available to any subscribed user, because it cannot predict with complete accuracy which files are needed at what point in time. Nor is there a reliable and secure method by which the server can be aware of certain information local to the

client computer that could be useful at stopping piracy. The process may be a rogue process intent on pirating the data, or it may be a secure process run from an executable provided by the ASP.

[0748] Aspects of the Invention

[0749] 1. Client-side fine-grained filtering of file accesses directed at remotely served files, for anti-piracy purposes. Traditional network filesystems permit or deny file access at the server side, not the client side. Here, the server provides blanket access to a given user to all the files that the user may need during the execution of an application, and makes more intelligent decisions about which accesses to permit or deny.

[0750] 2. Filtering of file accesses based on where the code for the process that originated the request is stored. Traditional file systems permit or deny file access usually based on the credentials of a user account or process token, not on where the code for the process resides. Here, a filesystem may want to take into account whether the code for the originating process resides in secure remote location or an insecure local location.

[0751] 3. Identification of crucial portions of served files and filtering file accesses depending on the portion targeted. The smallest level of granularity that traditional file systems can operate on is at the level of files, not at the level of the sections contained in the files (for example, whether or not data from a code section or a resource section is requested).

[0752] 4. Filtering of file accesses based on the surmised purpose of the file access, as determined by examining the program stack or flags associated with the request. Traditional file systems do not attempt to determine why a file access was issued before permitting or denying the access, e.g., whether the purpose is to copy the data or page in the data as code for execution.

[0753] 5. Filtering of file accesses based on the surmised purpose of the file access, as determined by examining a history of previous file accesses by the same process. Traditional file systems do not keep around histories of which blocks a given requestor had previously requested from a file. This history can be useful in seeing if the requests match a pattern that suggests a file copy is occurring as opposed to code execution.

[0754] Benefits of the Anti-Piracy Features of the Present Invention

[0755] This is an enabler technology that allows a programmer to build security into a certain type of application delivery system that would otherwise not be possible. Several companies are developing technology that allows an application to be served remotely, but executed locally. Current filesystems provide no way to protect the files that make up this application from being copied and thus pirated. The above techniques are tools that enable a filesystem to allow just those requests that will let the application run normally and block those that are the result of attempts to

pirate the application's code or data. This provides a competitive advantage to those software providers who use this technology, because piracy results in lost revenue and, by preventing this, piracy they can prevent this loss.

[0756] The techniques described herein were developed for the purpose of preventing the piracy of computer software programs that are served from a remote server, but executed on a local client. However, they can be used by any computer software security solution that would benefit from the ability to filter file accesses with more flexibility than currently provided by most filesystems.

[0757] When a filesystem receives a request, it must decide whether or not the request should be granted or denied for security reasons. If the target file is local, the filesystem makes the decision by itself, and if the target file is remote, it must ask the server to handle the request for it. The above techniques are ways in which the filesystem can gather more information about the request than it would ordinarily have. It can then use that information to improve the quality of its decisions. Traditional approaches, such as granting a currently logged-in user access to certain files and directories that are marked with his credentials, are not flexible enough for many situations. As for remote files, the server has only a limited amount of information about the client machine. The filesystem at the client side can make grant/deny decisions based on local information before ever asking the server, in order to provide a more intelligent layer of security.

[0758] For example, it may be desirable to allow the user to execute these files, but not copy them. It may be desirable to grant access to only certain processes run by the user, but not others, because it is judged that some processes to be more secure or well-behaved than others. And it may be desirable to allow the user to access only certain sections of these files and from only certain processes for certain periods of time. The above techniques are tools that are added to a filesystem to give it these abilities.

[0759] Overview of the Anti-Piracy Features of the Present Invention

[0760] With respect to FIG. 40, preventing piracy of remotely served, locally executed applications is shown. This figure illustrates the problem of software piracy in an application delivery system, and how it can be stopped using the techniques described in this section. The client computer 4001 is connected to a server 4009 run by an ASP 4007. The server 4009 provides access to application files 4008, out of which the application executable is run by the client 4001 locally on his machine. (This is Process #14002). However, the user can attempt to access and copy the application files to local storage 4009 on his machine, and thus be able to run them without authorization or give them to another person. But since all requests directed at the remote files 4006 must first pass through the local network filesystem, this filesystem can be enhanced 4005 to deny all such requests that it thinks are the result of an attempt at piracy.

[0761] Referring to FIG. 41, the filtering of accesses to remote application files, illustrating New Technique #1, as described above is shown. (Note: the client computer represented here and in all subsequent figures is part of the same client-server system as in FIG. 40, but the server/ASP diagram has been omitted to save space.) A user 4102 who

has been granted access to remotely served files 4106 representing an application is attempting to access these files. The local enhanced network filesystem 4103 is able to deny access to certain files 4105 and grant access to others 4104, for the purpose of protecting critical parts of the application from piracy.

[0762] With respect to FIG. 42, the filtering of accesses to remote files based on process code location, illustrating New Technique #2, as described above, is shown. Here there are two processes on the client computer. Process #14202 has been run from an executable file 4206 that is part of a remotely served application 4207, and process #24203 has been run from a local executable file 4204. They are both attempting to access a remote data file 4206 that is part of the served application 4207. The local enhanced network filesystem 4205 is denying Process #24203 access and granting Process #14202 access because Process #2's 4203 executable is stored locally, and thus is not secure, while Process #1's 4202 executable is provided by the server 4207, and thus can be vouched for.

[0763] Referring to FIG. 43, the filtering of accesses to remote files based on targeted file section, illustrating New Technique #3, as described above, is shown. Here there is a single local process 4302 that is attempting to read from a remotely served executable file 4307. The enhanced network filesystem 4304 is denying an attempt to read from the code section 4306 of the file 4307 while granting an attempt to read from a non-code section 4305 of the file 4307. This is useful when access to some part of the file must be allowed, but access to other parts should be denied to prevent piracy of the entire file.

[0764] With respect to FIG. 44, the filtering of accesses to remote files based on surmised purpose, illustrating New Technique #4 as described above, is shown. Here, two attempts to read from the code section 4407 of a remote executable file 4408 are being made from a process 4402 that was run from this file 4408. However, one request is denied because it originated 4406 from the process's code 4403 itself, while another is approved because it originated from code in the Virtual Memory Subsystem 4404. This prevents even a rogue remote process from attempting to pirate its own code, while allowing legitimate requests for the code to be completed.

[0765] Referring to FIG. 45, the filtering of accesses to remote files based on past access history, illustrating New Technique #5 as described above, is shown. Here, two processes 4502, 4503 run from a local executable 4504 are attempting to access a remote file 4508. The enhanced network filesystem 4507 keeps around a history of previous file accesses by these processes 4505, 4506, which it consults to make decisions about permitting/denying further accesses. Process #1's 4502 access attempt is granted, while Process #2's 4503 is denied, because the filesystem 4507 detected a suspicious pattern in Process #2's 4503 previous access history 4506.

[0766] Anti-Piracy Details of the Invention

[0767] Five anti-piracy embodiments are disclosed below that can be used by an ASP-installed network filesystem to combat piracy of remotely served applications. The ASP installs a software component on the client that is able to take advantage of local knowledge, e.g., which process on

the client originated a request for data, and permit or deny requests for remote files before sending the requests to the server. That is, a network filesystem is installed on the local user's computer that manages access to these remote files. All input/output requests to these files must pass through this filesystem, and if the filesystem determines that a given request is suspicious in some way, it has the freedom to deny it.

[0768] Anti-Piracy Embodiment #1

[0769] Client-side fine-grained filtering of file accesses directed at remotely served files, for anti-piracy purposes.

[0770] Referring again to FIG. 41, the approach of the first anti-piracy embodiment is that a software component 4102 executing locally on a client computer 4101 has available to it much more information about the state of this computer than does a server providing access to remote files. Thus, the server can filter access only on a much coarser level than can this client component. An ASP can take advantage of this by installing a network filesystem 4103 on the client computer that is designated to handle and forward all requests directed at files located on a given remote server. This filesystem 4103 examines each request, and either grants or denies it depending on whether the request is justifiable from a security perspective. It can use information such as the nature of the originating process, the history of previous access by the process, the section of the targeted file being requested, and so on, in order to make its decision.

[0771] The best way known of implementing this approach is to write a network redirector filesystem component 4103 for the operating system that the ASP's clients' machines will be running. This component will be installed, and will make visible to the system a path that represents the server on which the ASP's application files are stored. The local computer can now begin accessing these files, and the filesystem 4103 will be asked to handle requests for these files. On most operating systems, the filesystem 4103 will register dispatch routines to the system that handle common file operations such as open, read, write and close. When a local process 4102 makes a request of an ASP-served file, the OS calls one of these dispatch routines with the request. In the dispatch routine, the filesystem 4103 examines the request and decides whether to deny it or grant it. If granted, it will forward the request to the remote server and send back the response to the operating system.

[0772] Anti-Piracy Embodiment #2

[0773] Filtering of file accesses based on where the code for the process that originated the request is stored.

[0774] Referring again to FIG. 42, when a filesystem 4205 receives a request for access to a given file, the request always originates from a given process on the computer. By determining where the executable file that the process was run from is located, the network filesystem 4205 can make a more informed decision about the security risk associated with granting the request. For example, if the executable file 4204 is located on the local computer 4202, then it may contain any code whatsoever, code that may attempt to copy and store the contents of any remote files it can gain access to. The filesystem 4205 can reject requests from these processes as being too risky. However, if the executable file 4206 is being served by the ASP's remote server 4207, then the process can assume to be well-behaved, since it is under

the control of the ASP. The filesystem 4205 can grant accesses that come from these processes 4202 in confidence that the security risks are minimal.

[0775] The best way known of implementing this approach is to modify a network filesystem 4205 to determine the identity of the process that originated a relevant open, read, or write request for a remote file. On some OSes a unique process ID is embedded in the request, and on others, a system call can be made to get this ID. Then, this ID must be used to look up the pathname of the executable file from which the process was run. To do this, upon initialization the filesystem 4205 must have registered a callback that is invoked whenever a new process is created. When this callback is invoked, the pathname to the process executable and the new process ID are provided as arguments, data which the filesystem 4205 then stores in a data structure. This data structure is consulted while servicing a file request, in order to match the process ID that originated the request with the process's executable. Then the root of the pathname of that executable is extracted. The root uniquely identifies the storage device or remote server that provides the file. If the root specifies an ASP server that is known to be secure, as opposed to a local storage device that is insecure, then the request can be safely granted.

[0776] Anti-Piracy Embodiment #3

[0777] Identification of crucial portions of served files and filtering file access depending on the portion targeted.

[0778] Referring again to FIG. 43, a served application usually consists of many files. In order to steal the application, a pirate would have to copy at least those files that store the code for the application's primary executable, and perhaps other files as well. This leads to the conclusion that some files are more important than others, and that some portions of some files are most important of all. Ordinarily, the best solution would be to deny access to the primary executable file and its associated executables in its entirety, but this is not usually possible. In order to initially run the application, the filesystem 4304 must grant unrestricted access to some portions of the primary executable. In order to prevent piracy, the filesystem 4304 can grant access selectively to just those portions that are needed. Additionally, the running application 4302 itself does not usually need to read its own code section, but does need to read other sections for purposes such as resource loading. Therefore, additional security can be introduced by denying access to the code sections 4306 of ASP-served executables 4307 even to those executables themselves.

[0779] To implement this, modify a network filesystem's 4304 open file dispatch routine to detect when a remotely served executable 4307 is being opened. When this is detected, the executable file 4307 is examined to determine the offset and length of its code section 4306, and this information is stored in a data structure. On most OSes, executable files contain headers from which this information can be easily read. In the read and write dispatch routines, the network filesystem 4304 checks if the request is for a remote executable 4307, and if so, the offset and length of the code section 4306 of this executable 4307 is read from the data structure in which it was previously stored. Then the offset and length of the request are checked to see if they intersect the code section 4306 of this executable 4307. If so, the request can be denied.

[0780] Anti-Piracy Embodiment #4

[0781] Filtering of file accesses based on the surmised purpose of the file access, as determined by examining the program stack or flags associated with the request.

[0782] Referring again to FIG. 44, the approach of the fourth embodiment is that identical requests from the same process for a remotely served file can be distinguished based on the reason the request was issued. For example, on a computer with a virtual memory subsystem 4404, the VMS's own code will be invoked to page-in code for a process that attempts to execute code in pages that are not currently present. To do this, the VMS 4404 must issue a read request to the filesystem 4405 that handles the process' 4402 executable file 4408. Since this request is not for any ulterior purpose, such as piracy, and is necessary for the application to execute, the request should be granted. If the filesystem 4405 gets the originating process ID for such requests, the process whose code is being paged in will be known. However, this same process ID will also be returned for requests that originate as a result of an attempt by the process itself to read its own code (perhaps for the purpose of piracy). Many applications have loopholes that allow the user to execute a macro, for example, that reads and writes arbitrary files. If the filesystem 4405 simply filters requests based on process IDs, it will mistakenly allow users to pirate remotely served applications, as long as they can send the necessary reads and writes from within the remote application itself.

[0783] However, even if the process IDs are the same for two apparently identical requests, there are ways the filesystem 4405 can distinguish them. There are two known ways to do this in a manner relevant to combating anti-piracy. The way to implement the first method is to have the filesystem 4405, upon receiving a read request, check for the presence of the paging I/O flag that is supported by several operating systems. If this flag is not present, then the request did not come from the VMS 4404, but from the process itself 4403, and thus the request is risky and not apparently necessary for the application to run. If the flag is present though, the request almost certainly originated from the VMS 4404 for the purpose of reading in code to allow the process to execute. The request should be allowed.

[0784] Another way to make this same determination is to have the filesystem 4405 examine the program stack upon receiving a read request. In several operating systems, a process will attempt to execute code that resides in a virtual page regardless of whether the page is present or not. If the page is not present, a page fault occurs, and a structure is placed onto the stack that holds information about the processor's current state. Then the VMS 4404 gets control. The VMS 4404 then calls the read routine of the filesystem 4405 that handles the process's executable file to read this code into memory. The filesystem 4405 now reads backwards up the stack up to a certain point, searching for the presence of the structure that is placed on the stack as a result of a page fault. If such a structure is found, the execution pointer register stored in the structure is examined. If the pointer is a memory address within the boundary of the virtual memory page that is being paged in, then the filesystem 4405 knows the read request is legitimate.

[0785] Anti-Piracy Embodiment #5

[0786] Filtering of file accesses based on the surmised purpose of the file access, as determined by examining a history of previous file accesses by the same process.

[0787] Referring again to **FIG. 45**, if one looks at the series of file requests that are typically made as a result of attempting to copy an executable file, as opposed to those made in the course of executing that file, one can see certain patterns. The copy pattern is usually a sequence of sequentially ordered read requests, while the execution pattern tends to jump around a lot (as the result of code branches into non-present pages). A filesystem can be enhanced to keep around a history of requests made by specific processes on remotely served files. Then, for every subsequent request to such a file, the history for the originating process can be examined to check for certain patterns. If a file-copy pattern is seen, then the pirate may be attempting to steal the file, and the request should be denied. If an execution type pattern is seen, then the user is simply trying to run the application, and the request should be granted.

[0788] To implement this, a filesystem **4507** will tell the operating system, via an operating system call, upon initialization, to call it back whenever a new process is created. When it is called back, the filesystem **4507** will create a new data structure for the process that will store file access histories **4505**, **4506**. Then, in its read-file dispatch routines, the filesystem **4507** will determine the process ID of the originating process, and examine the process's access history **4505**, **4506**. It will only examine entries in that history **4505**, **4506** that refer to the file currently being requested. It will then run a heuristic algorithm that tries to determine if the pattern of accesses more closely resembles an attempted file copy than code execution. An effective algorithm is to simply see if the past *n* read requests to this file have been sequential, where *n* is some constant. If so, then the request is denied. If not, then the request is granted. In either case, an entry is made to the filesystem's process access history **4505**, **4506** that records the file name, offset, and length of the request made by that process to this file.

Conclusion

[0789] Although the present invention has been described using particular illustrative embodiments, it will be understood that many variations in construction, arrangement and use are possible within the scope of this invention. Other embodiments may use different network protocols, different programming techniques, or different heuristics, in each component block of the invention. Specific examples of variations include:

[0790] The proxy used in Proxy Caching could be anywhere in the Internet along the network path between a Client and the Server; and

[0791] Concurrent Requesting and Smart Requesting can be implemented in hardware instead of software.

[0792] A number of insubstantial variations are possible in the implementation of anti-piracy features of the invention. For example, instead of modifying the filesystem proper to provide anti-piracy features, a network proxy component can be placed on the client computer to filter network requests made by a conventional local network filesystem. These requests generally correspond to requests for remote

files made to the filesystem by a local process, and the type of filtering taught by the present invention can be performed on these requests. A filesystem filter component can also be written to implement these methods, instead of modifying the filesystem itself.

[0793] Although the invention is described herein with reference to the preferred embodiment, one skilled in the art will readily appreciate that other applications may be substituted for those set forth herein without departing from the spirit and scope of the present invention. Accordingly, the invention should only be limited by the Claims included below.

1. A process for converting a conventionally coded computer application program into a data set suitable for streamed delivery across a network from a server and concurrent execution on a client in a computer environment, comprising the steps of:

providing installation monitoring means for monitoring the installation process of said conventionally coded application program on a local computer system;

wherein said installation monitoring means monitors the modifications that said installation process makes to the system registry of said local computer system and records the system modification data;

wherein said installation monitoring means monitors and records any file modifications made by said installation process;

sorting said system modification data and said file modification data and removing duplicate entries;

parameterizing all of said local computer system's specific registry keys, value names, and values in said system modification data and said file modification data; and

providing data set creation means for processing said parameterized system modification data and said parameterized file modification data to create a data set suitable for streaming over said network.

2. The process of claim 1, wherein said data set creation means creates a runtime data set, said runtime data set consists of all regular application files and directories containing information about said regular application files.

3. The process of claim 2, wherein said data set creation means creates an initialization data set that is the first set of data streamed from said server to said client, said initialization data set prepares said client for streaming of said runtime data set.

4. The process of claim 2, wherein said directories contain lists of file names, file numbers, and the metadata associated with the files in a particular directory.

5. The process of claim 1, wherein said data set creation means creates a versioning table that contains a list of root file numbers and version numbers for tracking application patches and upgrades, and wherein each entry in said versioning table corresponds to one patch level of an application with a corresponding new root directory.

6. The process of claim 5, wherein said versioning table is sent to said client by said server, said client compares said versioning table with said client's root file number for the particular application program to find the necessary files required for a software upgrade or patch.

7. The process of claim 1, further comprising the step of:

providing a user interface that allows an operator to examine all changes made to said local computer system during said installation process and to edit said system modification data and said file modification data.

8. The process of claim 1, wherein said installation monitoring means monitors said application program as it runs and is being configured for a particular working environment on said local computer system and records common configurations of said application program thereby allowing said common configurations to be automatically duplicated on other client machines.

9. The process of claim 1, further comprising the step of:

program profiling means for capturing the sequence of file blocks being accessed during normal execution of said application program.

10. The process of claim 9, wherein said sequence of file blocks is used to pre-cache frequently used blocks on said client before said application program is first used by a user.

11. The process of claim 9, wherein said sequence of file blocks is used to optimize large directories of files on said client for faster file accesses.

12. The process of claim 9, wherein said sequence of file blocks is tied to specific user input and wherein said client pre-fetches file blocks based on user input to said application program.

13. The process of claim 1, wherein said installation monitoring means records the state of said local computer system before said installation process begins to give a more accurate picture of any modifications that are observed by said installation monitoring means.

14. An apparatus for converting a conventionally coded computer application program into a data set suitable for streamed delivery across a network from a server and concurrent execution on a client in a computer environment, comprising:

installation monitoring means for monitoring the installation process of said conventionally coded application program on a local computer system;

wherein said installation monitoring means monitors the modifications that said installation process makes to the system registry of said local computer system and records the system modification data;

wherein said installation monitoring means monitors and records any file modifications made by said installation process;

a module for sorting said system modification data and said file modification data and removing duplicate entries;

a module for parameterizing all of said local computer system's specific registry keys, value names, and values in said system modification data and said file modification data; and

data set creation means for processing said parameterized system modification data and said parameterized file modification data to create a data set suitable for streaming over said network.

15. The apparatus of claim 14, wherein said data set creation means creates a runtime data set, said runtime data set consists of all regular application files and directories containing information about said regular application files.

16. The apparatus of claim 15, wherein said data set creation means creates an initialization data set that is the first set of data streamed from said server to said client, said initialization data set prepares said client for streaming of said runtime data set.

17. The apparatus of claim 15, wherein said directories contain lists of file names, file numbers, and the metadata associated with the files in a particular directory.

18. The apparatus of claim 14, wherein said data set creation means creates a versioning table that contains a list of root file numbers and version numbers for tracking application patches and upgrades, and wherein each entry in said versioning table corresponds to one patch level of an application with a corresponding new root directory.

19. The apparatus of claim 18, wherein said versioning table is sent to said client by said server, said client compares said versioning table with said client's root file number for the particular application program to find the necessary files required for a software upgrade or patch.

20. The apparatus of claim 14, further comprising:

a user interface that allows an operator to examine all changes made to said local computer system during said installation process and to edit said system modification data and said file modification data.

21. The apparatus of claim 14, wherein said installation monitoring means monitors said application program as it runs and is being configured for a particular working environment on said local computer system and records common configurations of said application program thereby allowing said common configurations to be automatically duplicated on other client machines.

22. The apparatus of claim 14, further comprising:

program profiling means for capturing the sequence of file blocks being accessed during normal execution of said application program.

23. The apparatus of claim 22, wherein said sequence of file blocks is used to pre-cache frequently used blocks on said client before said application program is first used by a user.

24. The apparatus of claim 22, wherein said sequence of file blocks is used to optimize large directories of files on said client for faster file accesses.

25. The apparatus of claim 22, wherein said sequence of file blocks is tied to specific user input and wherein said client pre-fetches file blocks based on user input to said application program.

26. The apparatus of claim 14, wherein said installation monitoring means records the state of said local computer system before said installation process begins to give a more accurate picture of any modifications that are observed by said installation monitoring means.

27. A program storage medium readable by a computer, tangibly embodying a program of instructions executable by the computer to perform method steps for converting a conventionally coded computer application program into a data set suitable for streamed delivery across a network from a server and concurrent execution on a client in a computer environment, comprising the steps of:

providing installation monitoring means for monitoring the installation process of said conventionally coded application program on a local computer system;

wherein said installation monitoring means monitors the modifications that said installation process makes to the system registry of said local computer system and records the system modification data;

wherein said installation monitoring means monitors and records any file modifications made by said installation process;

sorting said system modification data and said file modification data and removing duplicate entries;

parameterizing all of said local computer system's specific registry keys, value names, and values in said system modification data and said file modification data; and

providing data set creation means for processing said parameterized system modification data and said parameterized file modification data to create a data set suitable for streaming over said network.

28. The method of claim 27, wherein said data set creation means creates a runtime data set, said runtime data set consists of all regular application files and directories containing information about said regular application files.

29. The method of claim 28, wherein said data set creation means creates an initialization data set that is the first set of data streamed from said server to said client, said initialization data set prepares said client for streaming of said runtime data set.

30. The method of claim 28, wherein said directories contain lists of file names, file numbers, and the metadata associated with the files in a particular directory.

31. The method of claim 27, wherein said data set creation means creates a versioning table that contains a list of root file numbers and version numbers for tracking application patches and upgrades, and wherein each entry in said versioning table corresponds to one patch level of an application with a corresponding new root directory.

32. The method of claim 31, wherein said versioning table is sent to said client by said server, said client compares said versioning table with said client's root file number for the particular application program to find the necessary files required for a software upgrade or patch.

33. The method of claim 27, further comprising the step of:

providing a user interface that allows an operator to examine all changes made to said local computer system during said installation process and to edit said system modification data and said file modification data.

34. The method of claim 27, wherein said installation monitoring means monitors said application program as it runs and is being configured for a particular working environment on said local computer system and records common configurations of said application program thereby allowing said common configurations to be automatically duplicated on other client machines.

35. The method of claim 27, further comprising the step of:

program profiling means for capturing the sequence of file blocks being accessed during normal execution of said application program.

36. The method of claim 35, wherein said sequence of file blocks is used to pre-cache frequently used blocks on said client before said application program is first used by a user.

37. The method of claim 35, wherein said sequence of file blocks is used to optimize large directories of files on said client for faster file accesses.

38. The method of claim 35, wherein said sequence of file blocks is tied to specific user input and wherein said client pre-fetches file blocks based on user input to said application program.

39. The method of claim 27, wherein said installation monitoring means records the state of said local computer system before said installation process begins to give a more accurate picture of any modifications that are observed by said installation monitoring means.

* * * * *

(C) Evidence for Claims 1-27 – Entered by Examiner

The following items (1) – (33) listed below are hereby entered as evidence entered by the Examiner. Also listed for each item is where said evidence was entered into the record by the Examiner.

- (1) Copy of US Patent Number 6631417 (“Balabine”). This evidence was entered into the record by the Examiner on page 2 paragraph 2 of the Office Action mailed 10/06/2004, on IDS sheet included in the Office Action mailed 10/06/2004.
- (2) Copy of US Patent Number 6385643 (“Jacobs et al.”). This evidence was entered into the record by the Examiner on page 2 paragraph 2 of the Office Action mailed 10/06/2004, on IDS sheet included in the Office Action mailed 10/06/2004.
- (3) Copy of US Patent Number 6751671 (“Urien”). This evidence was entered into the record by the Examiner on page 2 paragraph 2 of the Office Action mailed 01/31/2005, on IDS sheet included in the Office Action mailed 01/31/2005.
- (4) Copy of US Patent Number 6795851 (“Noy”). This evidence was entered into the record by the Examiner on page 2 paragraph 2 of the Office Action mailed 08/12/2005, and on IDS sheet included in the Office Action mailed 08/12/2005.
- (5) Copy of US Patent Number 6236999 (“Jacobs et al.”). This evidence was entered into the record by the Examiner on page 2 paragraph 2 of the Office Action mailed 08/12/2005, and on IDS sheet included in the Office Action mailed 08/12/2005.
- (6) Copy of US Patent Application Number 20020083183 (“Pujare et al.”). This evidence was entered into the record by the Examiner on page 4 paragraph 5 of the Office Action mailed 08/12/2005, and on IDS sheet included in the Office Action mailed 08/12/2005.

- (7) Copy of US Patent Application Number 20020083183 ("Pujare et al."). This evidence was entered into the record by the Examiner on page 4 paragraph 5 of the Office Action mailed 08/12/2005, and on IDS sheet included in the Office Action mailed 08/12/2005.
- (8) Copy of US Patent Application Number 20030009571 ("Bavadekar"). This evidence was entered into the record by the Examiner on page 4 paragraph 3 of the Office Action mailed 03/23/2006, on IDS sheet included in the Office Action mailed 03/23/2006, and on IDS sheet included in the Office Action mailed 08/12/2005.
- (9) Copy of US Patent Number 6112246 ("Horbal et al."). This evidence was entered into the record by the Examiner on Notice of References Cited line C of the Office Action mailed 10/06/2004.
- (10) Copy of US Patent Number 6581088 ("Jacobs et al."). This evidence was entered into the record by the Examiner on Notice of References Cited line D of the Office Action mailed 10/06/2004.
- (11) Copy of US Patent Number 5999979 ("Vallanki et al."). This evidence was entered into the record by the Examiner on Notice of References Cited line A of the Office Action mailed 01/31/2005.
- (12) Copy of US Patent Number 5870549 ("Bobo, II"). This evidence was entered into the record by the Examiner on page 4 paragraph 13 of the Office Action mailed 01/31/2005, on IDS sheet included in the Office Action mailed 01/31/2005.
- (13) Copy of US Patent Number 6658463 ("Dillon et al."). This evidence was entered into the record by the Examiner on page 4 paragraph 13 of the Office Action mailed 01/31/2005, on IDS sheet included in the Office Action mailed 01/31/2005.

- (14) Copy of US Patent Number 6128653 ("del Val et al."). This evidence was entered into the record by the Examiner on Notice of References Cited line E of the Office Action mailed 01/31/2005.
- (15) Copy of US Patent Number 6795848 ("Border et al."). This evidence was entered into the record by the Examiner on Notice of References Cited line F of the Office Action mailed 01/31/2005.
- (16) Copy of US Patent Application Number 200291763 ("Shah et al."). This evidence was entered into the record by the Examiner on Notice of References Cited line D of the Office Action mailed 08/12/2005.
- (17) Copy of US Patent Number 6412009 ("Erickson et al."). This evidence was entered into the record by the Examiner on Notice of References Cited line E of the Office Action mailed 08/12/2005.
- (18) Copy of US Patent Application Number 20030009571 ("Bavadekar"). This evidence was entered into the record by the Examiner on page 4 paragraph 3 of the Office Action mailed 03/23/2006, on IDS sheet included in the Office Action mailed 03/23/2006, and on IDS sheet included in the Office Action mailed 08/12/2005.
- (19) Copy of US Patent Application Number 2002161904 ("Tredoux et al."). This evidence was entered into the record by the Examiner on Notice of References Cited line G of the Office Action mailed 08/12/2005.
- (20) Copy of US Patent Application Number 2003182431 ("Sturniolo et al."). This evidence was entered into the record by the Examiner on Notice of References Cited line B of the Office Action mailed 03/23/2006.

- (21) Copy of US Patent Number 6941377 ("Diamant et al."). This evidence was entered into the record by the Examiner on Notice of References Cited line C of the Office Action mailed 03/23/2006.
- (22) Copy of NPL - Layer 4+ Switching with QOS Support for RTP and HTTP; Harbaum, T.; IEEE 1999. This evidence was entered into the record by the Examiner on Notice of References Cited line U of the Office Action mailed 10/06/2004.
- (23) Copy of NPL - Optimizing TCP Forwarder Performance; Spatscheck, O; IEEE/ACM transactions on Networking, Vol. 8, No. 2, APRIL 2000. This evidence was entered into the record by the Examiner on Notice of References Cited line V of the Office Action mailed 10/06/2004.
- (24) Copy of NPL - Kernal Mechanisms for Service Differentiation in Overloaded Web Servers; Voigt, Tewari, Freimuth (2001); www.sics.se/~thiemo/usenix01.ps. This evidence was entered into the record by the Examiner on Notice of References Cited line W of the Office Action mailed 10/06/2004.
- (25) Copy of NPL - TRIAD: A New Next-Generation Internet Architecture - Cheriton, Gritter (2000); www-dsg.stanford.edu/triad/triad.ps.gz. This evidence was entered into the record by the Examiner on Notice of References Cited line X of the Office Action mailed 10/06/2004.
- (26) Copy of NPL - Distributed Packet Rewriting and its Application to Scalable Server Architectures; Bestavros, A; Crovella, M.; Liu, J; Martin, D.; (Oct 1998); www.cs.bu.edu/faculty/best/res/papers/icnp98.ps. This evidence was entered into the record by the Examiner on Notice of References Cited line U of the Office Action mailed 01/31/2005.

- (27) Copy of NPL - Memex: A browsing assistant for collaborative archiving and mining of surf trails; Chakrabarti, S; Srivastava, S; Subramanyam, M; Tiwari, M; (2000); www.vidb.org/conf/2000/P603.pdf. This evidence was entered into the record by the Examiner on Notice of References Cited line V of the Office Action mailed 01/31/2005.
- (28) Copy of NPL - APNNed goes Internet; Kemper, P., Tepper, C.; IS4-www.infomatik.uni-dortmund.de/QM/MA/pk/publication_ps_files/awpn01.ps.gz. This evidence was entered into the record by the Examiner on Notice of References Cited line W of the Office Action mailed 01/31/2005.
- (29) Copy of NPL - Lightweight, Dynamic and Programmable Virtual Private Networks; Isaacs, R; (2000) www.cl.cam.ac.uk/Research/SRG/netos/ncam/docs/public/openarch00.ps.gz. This evidence was entered into the record by the Examiner on Notice of References Cited line U of the Office Action mailed 08/12/2005.
- (30) Copy of NPL - Experiences from extending a legacy system with CORBA components; COT/4-08-V1.3; www.cit.dk/COT/reports/reports/Case4/08/cot-4-08.pdf. This evidence was entered into the record by the Examiner on Notice of References Cited line V of the Office Action mailed 08/12/2005.
- (31) Copy of NPL - Transparent Caching of Web Services for Mobile Devices; Elbashir, K; Multi-Agent bistrica.uask.ca/madmuc/Pubs/kamal880.pdf. This evidence was entered into the record by the Examiner on Notice of References Cited line W of the Office Action mailed 08/12/2005.
- (32) Copy of NPL - How to Turn a GSM SIM into a Web Server - Projecting Mobile Trust onto the World Wide Web; Guthery; Kehr; Posegga; www.teco.edu/~posegga/papers/WebSIM_Cards_Submission.pdf. This evidence was

entered into the record by the Examiner on Notice of References Cited line V of the Office
Action mailed 03/23/2006.

(33) Copy of NPL - A Database Computer Attacks for the Evaluation of Intrusion Detection
Systems; Kendall, K. (1999) www.kkendall.org/files/thesis/krkthesis.pdf. This evidence
was entered into the record by the Examiner on Notice of References Cited line U of the
Office Action mailed 03/23/2006.

Copies of all References follows.

//



US006112246A

United States Patent [19]**Horbal et al.**[11] **Patent Number:** **6,112,246**[45] **Date of Patent:** **Aug. 29, 2000**

[54] **SYSTEM AND METHOD FOR ACCESSING INFORMATION FROM A REMOTE DEVICE AND PROVIDING THE INFORMATION TO A CLIENT WORKSTATION**

5,528,219 6/1996 Frohlich et al. .
5,598,521 1/1997 Kilgore et al. .
5,664,101 9/1997 Picache .
5,794,032 8/1998 Leyda 713/2
5,805,442 9/1998 Crater et al. 364/138

[76] Inventors: **Mark T. Horbal**, 32802 Fowler Cir.,
Warrenville, Ill. 60555; **Randal J. King**, 3 S. 947 Thornapple Tree Rd.,
Sugar Grove, Ill. 60554

Primary Examiner—Zarni Maung
Attorney, Agent, or Firm—Banner & Witcoff, Ltd.

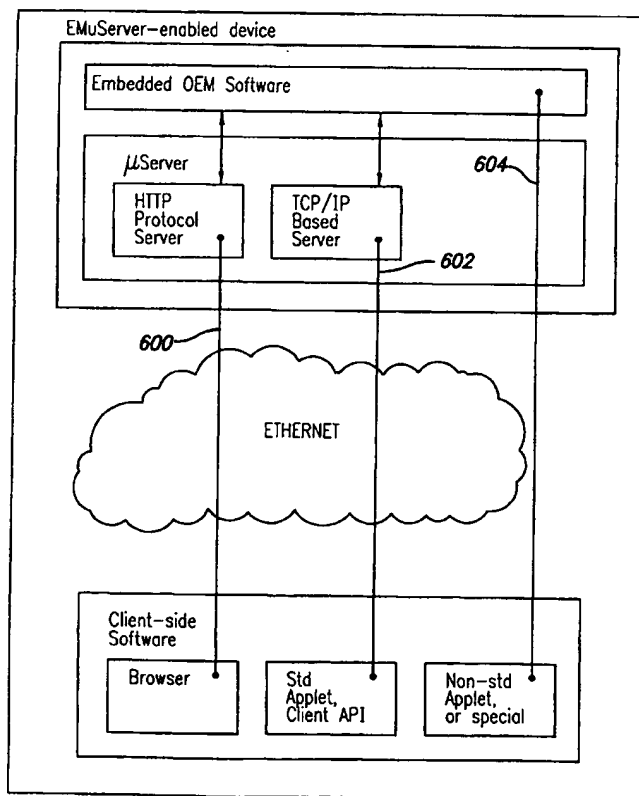
[57] **ABSTRACT**

A micro-server adapted to be embedded into a piece of industrial machinery, an automobile, a consumer product, and the like, for publishing information, possibly in the form of web pages, about the device into which the micro-server is embedded or with which it is associated and/or for controlling a micro-server equipped device from a possibly remote client. The information may be published such that it is accessible using a standard web-browser. Other suitable protocols could also be used. The micro-server is capable of interfacing with a device to access information from the device, such as control or maintenance information. The micro-server can then organize and format that information compatible with a communication protocol in preparation for publishing the information. The micro-server conveniently abstracts from the first device the details of the communication protocol used to publish the information.

[21] Appl. No.: 09/176,993

[22] Filed: **Oct. 22, 1998**[51] **Int. Cl.**⁷ **G08C 15/06**[52] **U.S. Cl.** **709/230**[58] **Field of Search** 364/131, 138;
345/333-335; 709/203, 217, 219, 223, 224,
230[56] **References Cited****U.S. PATENT DOCUMENTS**

4,860,216 8/1989 Linsenmayer .
4,901,218 2/1990 Cornwell .
5,428,555 6/1995 Starkey et al. .
5,472,347 12/1995 Nordenstrom et al. .
5,512,890 4/1996 Everson, Jr. et al. .

35 Claims, 17 Drawing Sheets

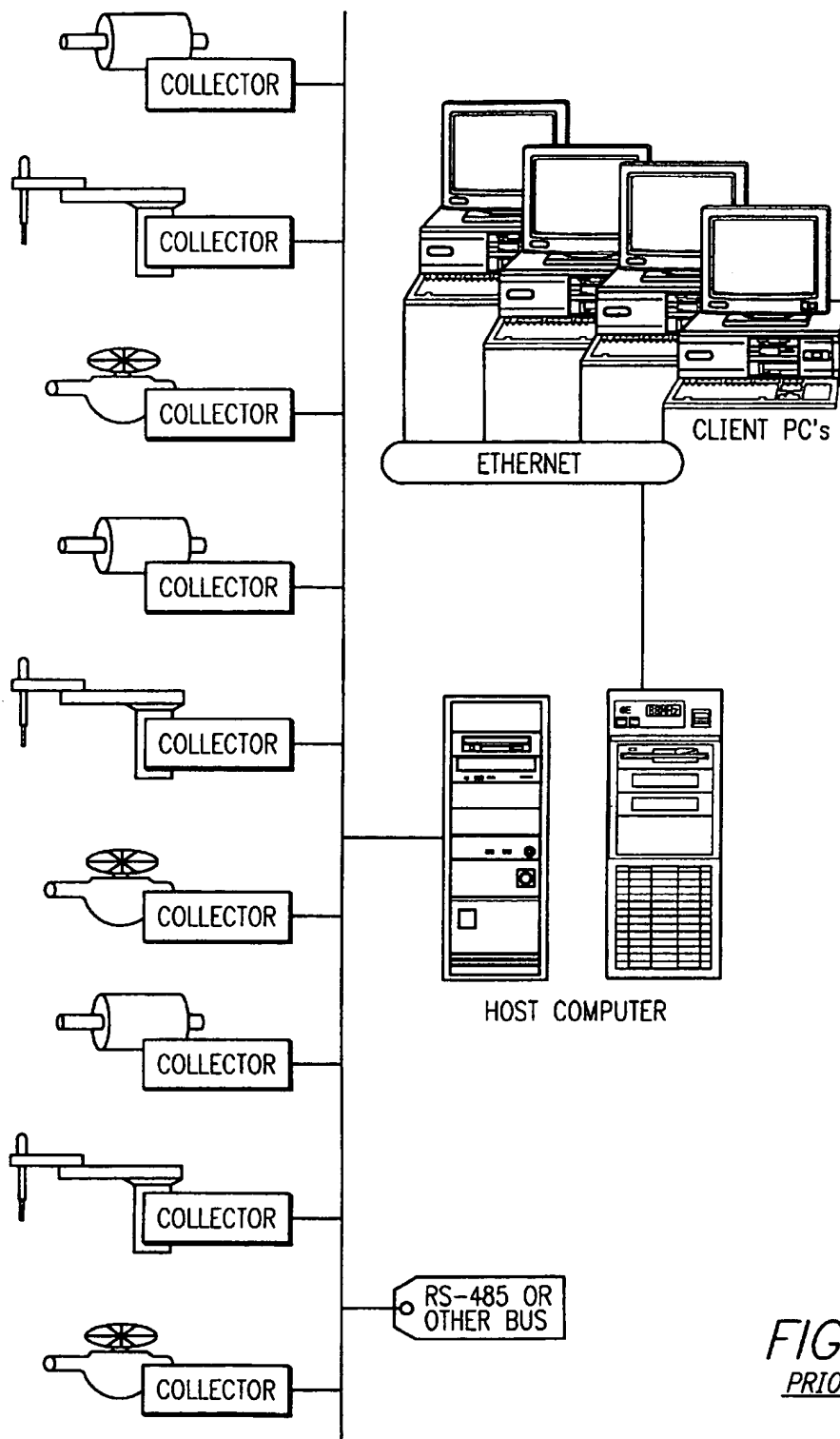


FIG. 1
PRIOR ART

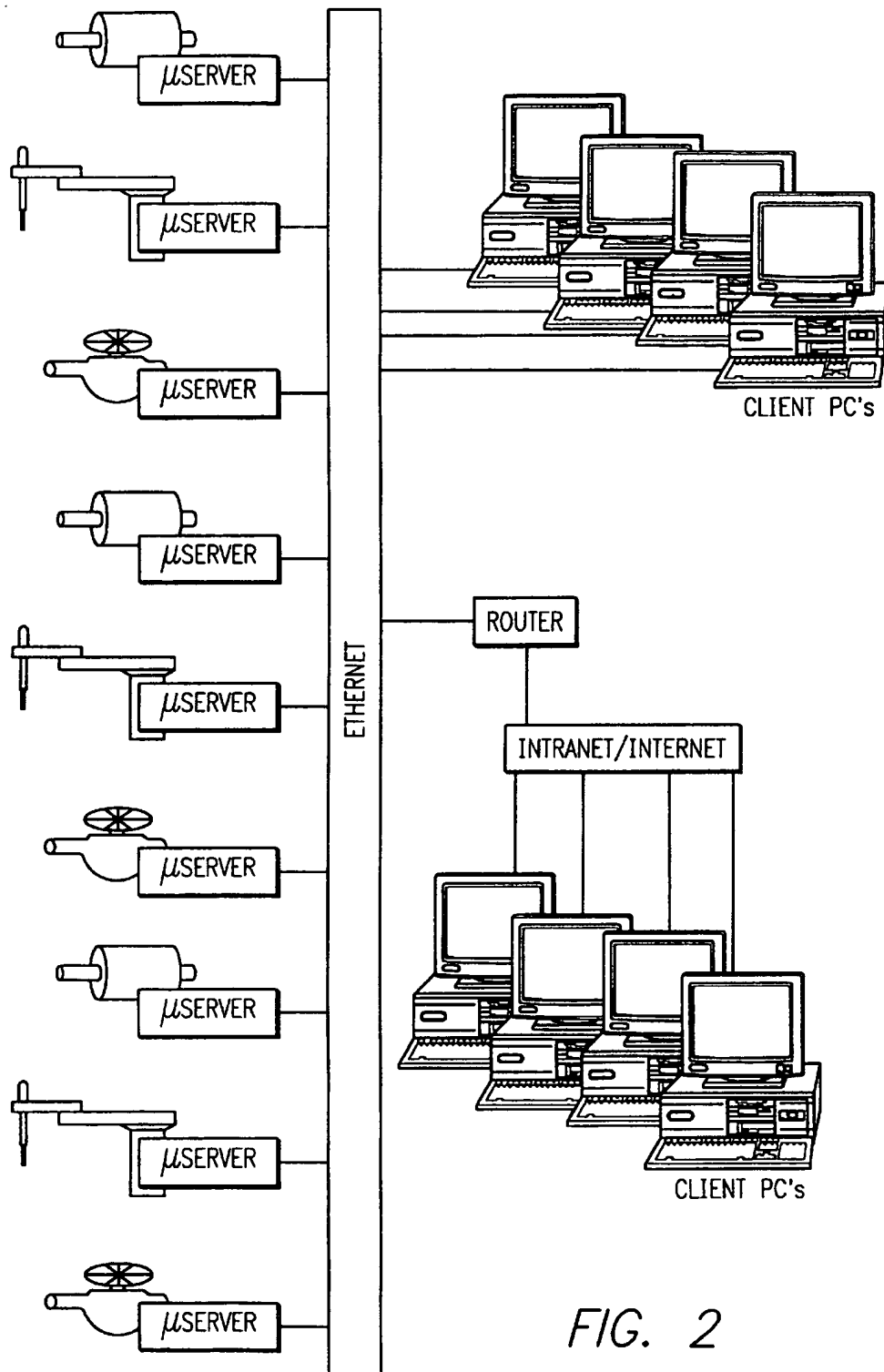
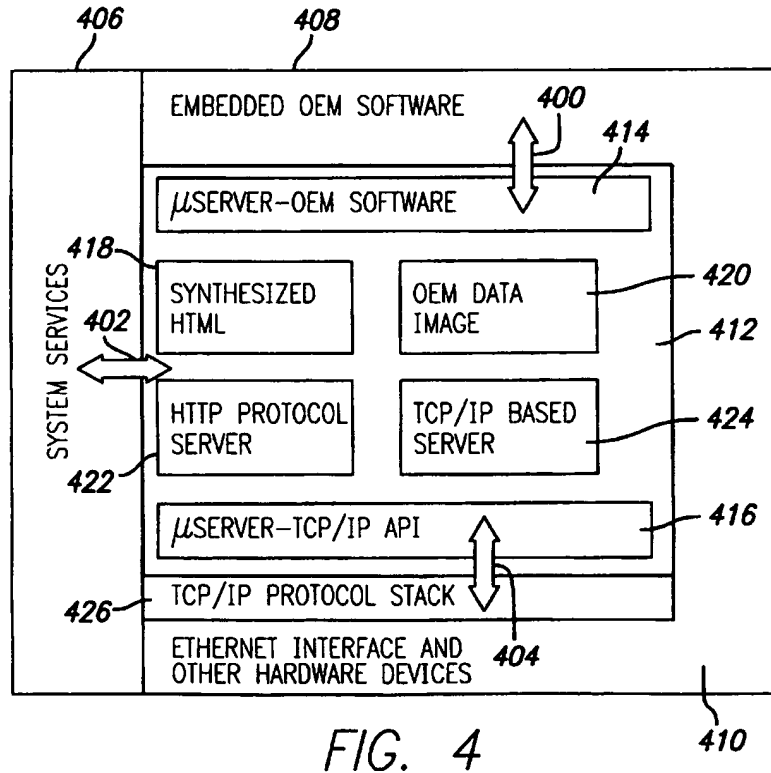
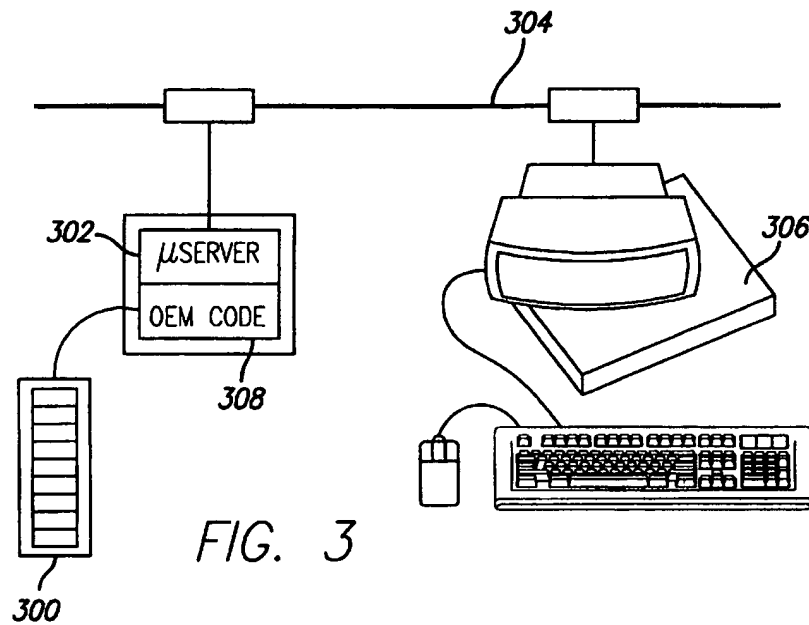


FIG. 2



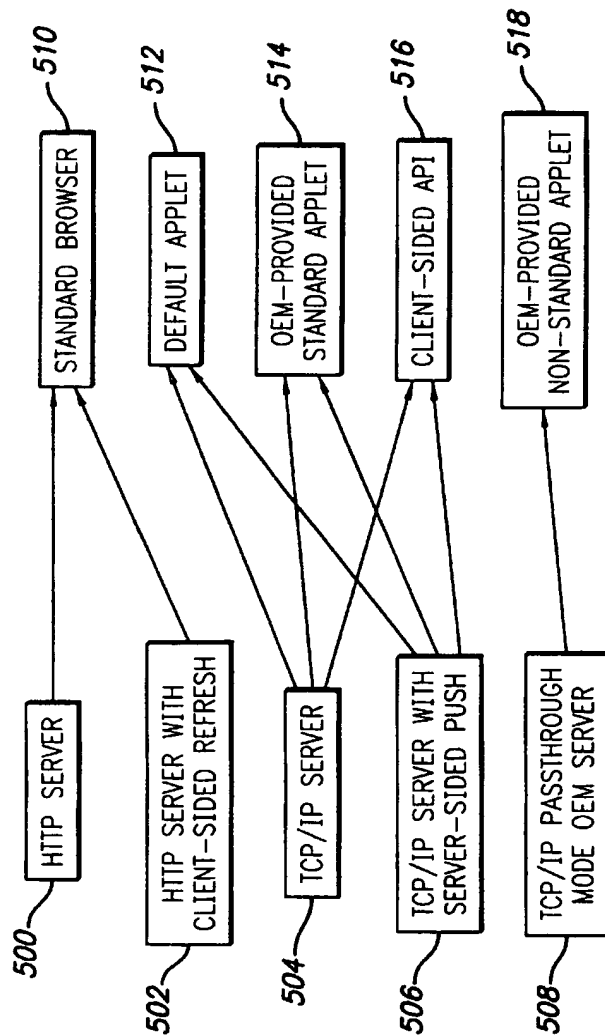


FIG. 5

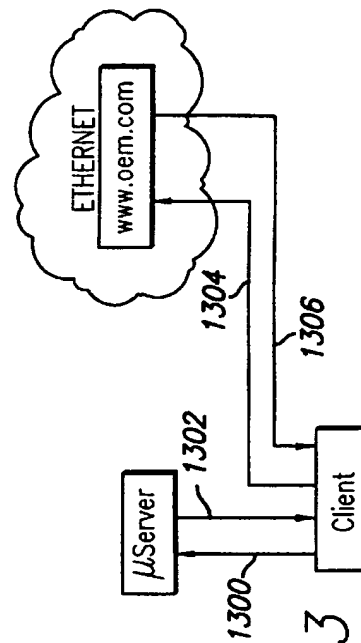


FIG. 13

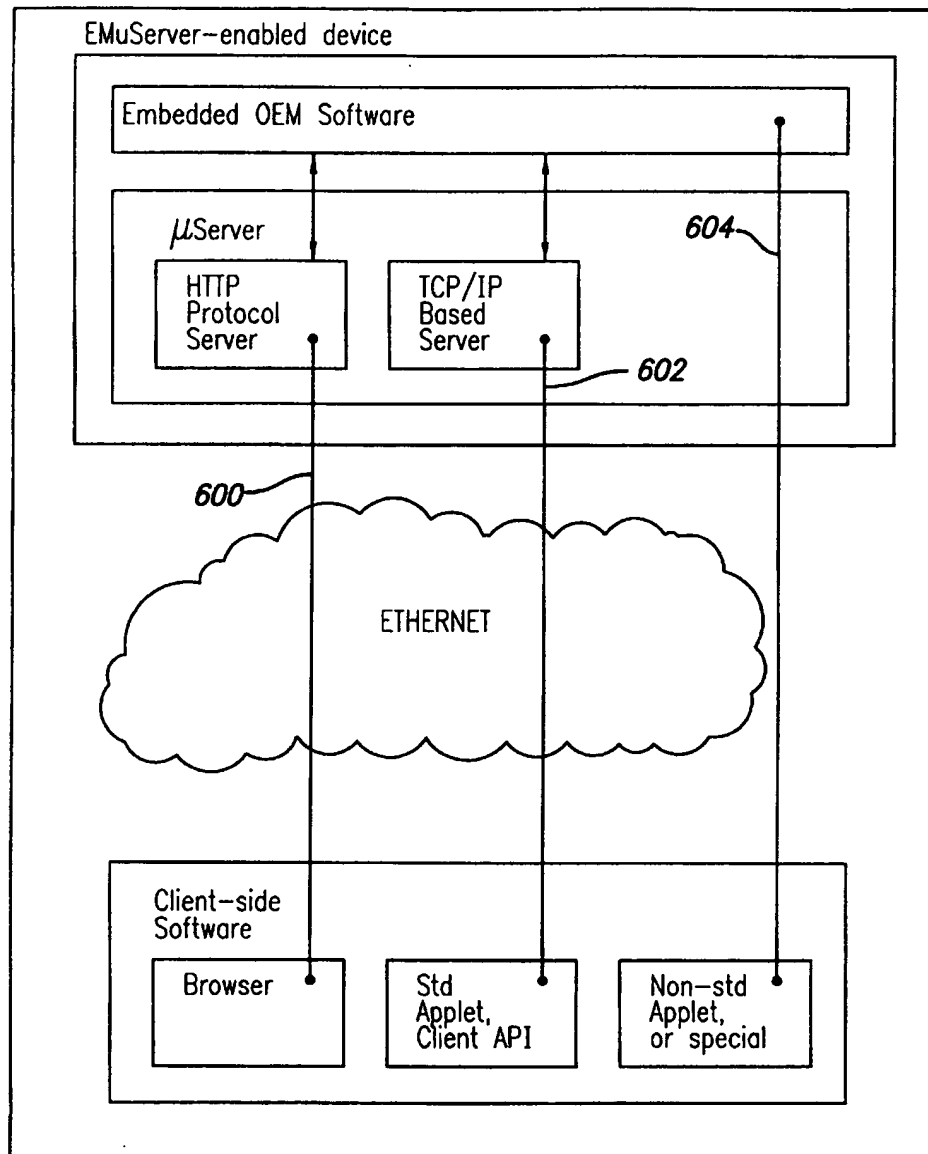


FIG. 6

FIG. 7

DAQ 37-Netscape

File Edit View Go Communication Help

Back Forward Reload Home Search Netscape Print Security Stop

Micro Server Release 1.42

Auto Refresh: off

Now: 4.16.98 14:07:23

Upd: 4.16.98 14:07:01

Mfg. Name V Technology Group Incorporated

Mfg. Desc. LubeMonDAQ Node

Mfg. Model: DAQ2.1

User Desc: Paint Shop Staging

User Id: CONVEYOR P156

User Loc: Bldg 5.A23

Read Control Maint Admin Discover

Parameter Values

Id	Parameters	Unit	Value	Alrm	Norm	Low	High
D0001	Drive Current	A	50.70		100.00	60.00	120.00
D0002	Drive Voltage	V	242.56		240.00	230.00	250.00
D0003	Drive Power	HP	12.12		13.00	9.00	15.00
D0004	Take-up Pressure	PSI	57.77	●	70.00	60.00	90.00
D0005	Reducer Temperature	°F	95.45		90.00	45.00	180.00
D0006	Chain Velocity	ft/min	60.02		50.00	20.00	80.00
D0007	Chain Growth	in	101.93		90.00	20.00	180.00
D0008	Chain Growth	%	0.015		90.00	20.00	180.00
D0009	Lubricant Level	%	56.93		50.00	10.00	

FIG. 8

DAQ 37-Netscope

File Edit View Go Communication Help

Back Forward Reload Home Search Netscope Print Security Stop

Micro Server Release 1.42

Auto Refresh: off

Now: 4.16.98 14:07:23

Upd: 4.16.98 14:07:01

Mfg. Name V Technology Group Incorporated

Mfg. Desc. LubeMonDAQ Node

Mfg. Model DAQ2.1

User Desc: Paint Shop Staging

User Id: CONVEYOR P156

User Loc: Bldg 6.A23

Read

Control

Maint

Admin

Discover

Write Parameter Values

Id	Parameters	Unit	Min	Norm	Max	New Value	Enter	Clear
D0010	Chain Pitch	in	3.00	4.00	6.00	4.00	Enter	Clear
D0011	Chain Velocity	ft/min	20.00	50.00	95.00	50.00	Enter	Clear
D0012	RUN Signal Present	...	no	yes	yes	yes	Enter	Clear
D0013	Chain Start Configuration	...	1	1	8	1	Enter	Clear
D0014	Digital Filter Order	...	1	5	20	5	Enter	Clear

FIG. 9

DAQ 37-Netscape

File Edit View Go Communication Help

Back Forward Reload Home Search Netscape Print Security Stop

Micro Server Release 1.42

V Technology Group Incorporated

LubeMonDAQ Node

Read

Control

Maint

Admin

Discover

Mfg. Name: V Technology Group Incorporated

Mfg. Desc: LubeMonDAQ Node

Mfg. Model: DAQ2.1

Mfg. Date: 97.11.02

Serial No: 98112344

Mfg. ID: 0078453295

Now: 4.16.98 14:07:23

Maintenance Log and User Entry

Id	Time_stamp	Type	Entry
MA0001	98.01.01 00:34:51	I	Reboot
MA0002	98.01.06 13:02:22	U	Performed calibration of onlog channels,adjusted CGM by J.B.
MA0004	98.01.06 15:17:04	U	User initiated selftest-PASSED
MA0005	98.01.06 15:17:04	I	Reboot
MA0006	98.02.28 15:17:06	E	A/D read timeout failure on channel 6
MA0007	98.04.23 03:24:56	I	Reboot
MA0008	<current timestamp>	U	Enter Text Here

Enter Clear

FIG. 10

DAQ 37-Netscape

File Edit View Go Communication Help

Back Forward Reload Home Search Netscape Print Security Stop

Maintenance Log and User Entry

Id	Time stamp	Type	Entry
MA0001	98.01.01 00:34:51	I	Reboot
MA0002	98.01.06 13:02:22	U	Performed calibration of analog channels, adjusted CGM by J.B.
MA0004	98.01.06 15:17:04	U	User initiated selftest-PASSED
MA0005	98.01.06 15:17:04	I	Reboot
MA0006	98.02.28 15:17:06	E	A/D read timeout failure on channel 6
MA0007	98.04.23 03:24:56	I	Reboot
MA0008	<current timestamp>	U	Enter Text Here

I=Information, U=User Entry, E=Error

Maintenance Functions

Self Test Performs self test now

M.Server Auxiliary Maintenance Server (not configured)

Mfg's Site Manufacturer's web site: <http://www.vtechnology.com>

Mail E-mail technical support

Docum Access Documentation

Enter Enter Clear

FIG. 11

DAQ 37--Netscape

File Edit View Go Communication Help

Back Forward Reload Home Search Netscape Print Security Stop

Micro Server Release 1.42

Mfg. Name V Technology Group Incorporated

Mfg. Desc. LubeMonDAQ Node

Mfg. Model DAQ2.1

User Desc: Paint Shop Staging

User Id: CONVEYOR P156

User Loc: Bldg 6.A23

Now: 4.16.98 14:07:23

Configuration Parameter Values

Id	Time stamp	Value	New Value	Enter	Clear
TSANITY	Sanity xmt interval trigger(sec)	20	30.00	Enter	Clear
TPCNTCHG	Pcnt change xmt interval trigger(1-100%)	0	20.00	Enter	Clear
TAUTOREF	Client-side Auto_refresh(0=off)	OFF	1	Enter	Clear
TALARM	Alarm based xmt trigger(0=off)	OFF	1	Enter	Clear
CDNSSVR	DNS Server IP address	121.045.067.044	121.045.067.044	Enter	Clear
CPUSHIPO	Push subscriber 0 IP address	034.067.089.012	034.067.089.012	Enter	Clear

FIG. 12

DAQ 37-Netscape

File Edit View Go Communication Help

Back Forward Reload Home Search Netscape Print Security Stop

Mfg. Name V Technology Group Incorporated

Mfg. Desc. LubeMonDAQ Node

Mfg. Model DAQ2.1

User Desc: Paint Shop Staging

User Id: CONVEYOR P156

User Loc: Bldg 6.A23

Micro Server Release 1.42

Auto Refresh: OFF

Now: 4.16.98 14:07:23

Upd: 4.16.98 14:07:01

Read

Control

Maint

Admin

Discover

Node Discovery Information

Id	Description	Value
MFGNAME	Manufacturer Name	V Technology Group
MFGADRS0	Manufacturer address 0	2001 S. Stoughton Road
MFGADRS1	Manufacturer address 1	Madison, Wisconsin 53716
MFGADRS2	Manufacturer address 2	USA
MFGADRS3	Manufacturer address 3	
MFGTEL	Manufacturer telephone number	(001 608)221.1100
MFGFAX	Manufacturer fax number	(001 608)221.1100
MFGEMAIL	Manufacturer tech support Email address	techsupport@vtechnology.com

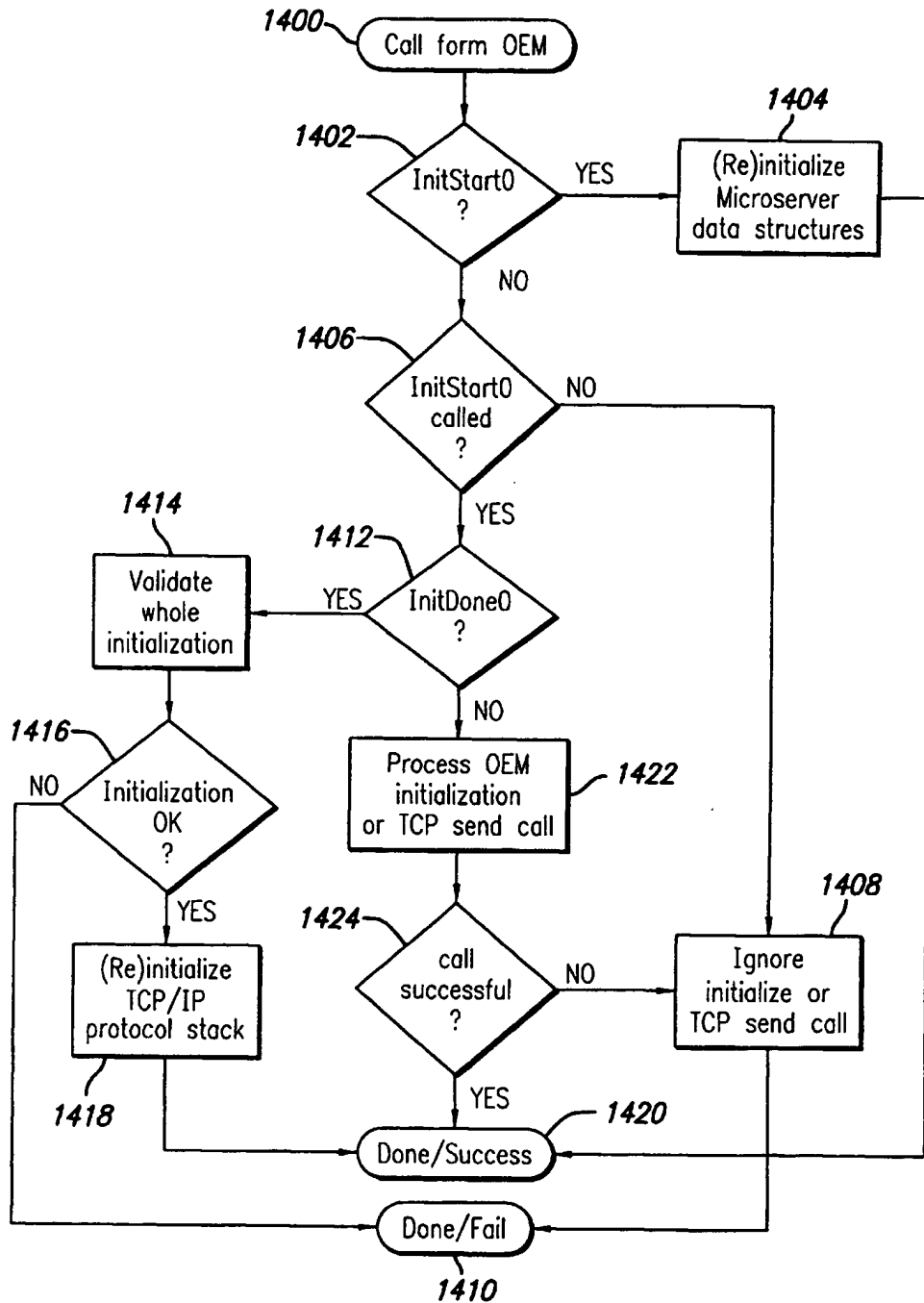


FIG. 14

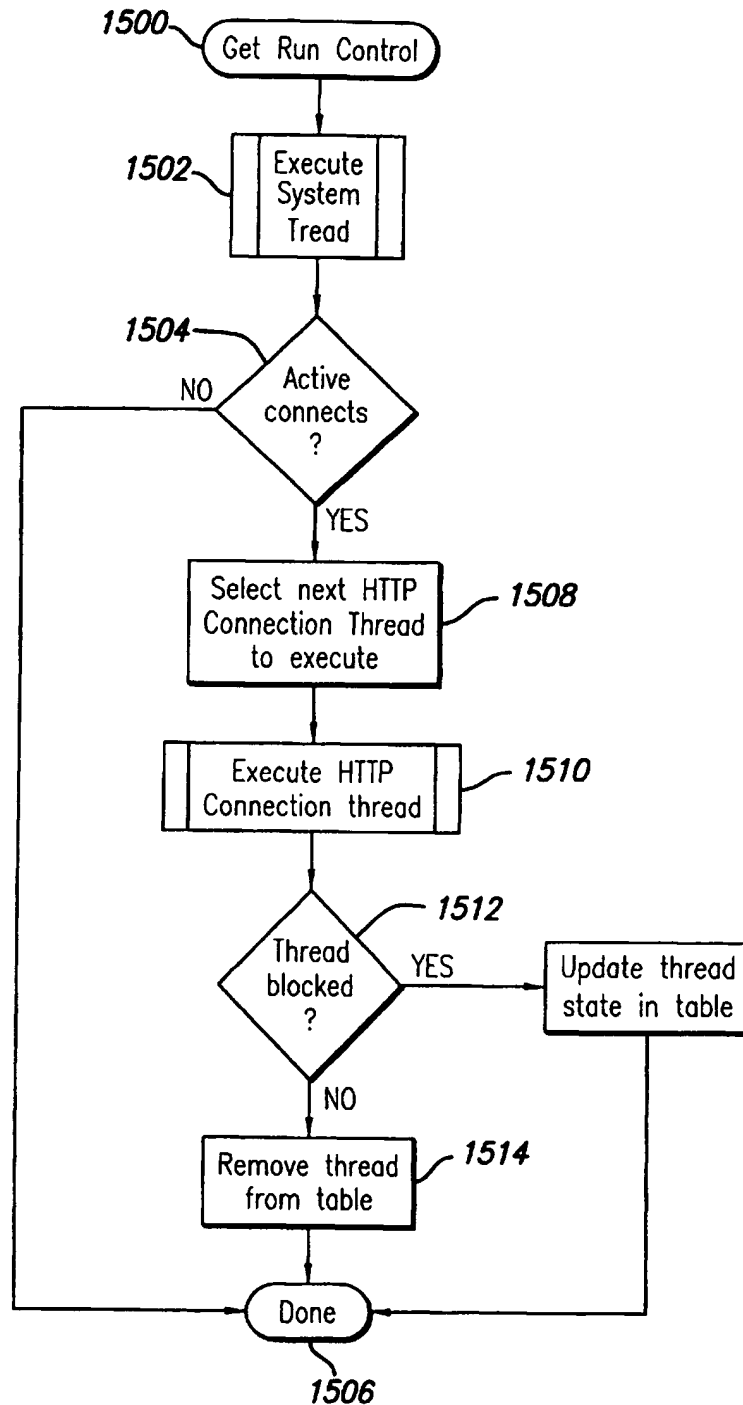
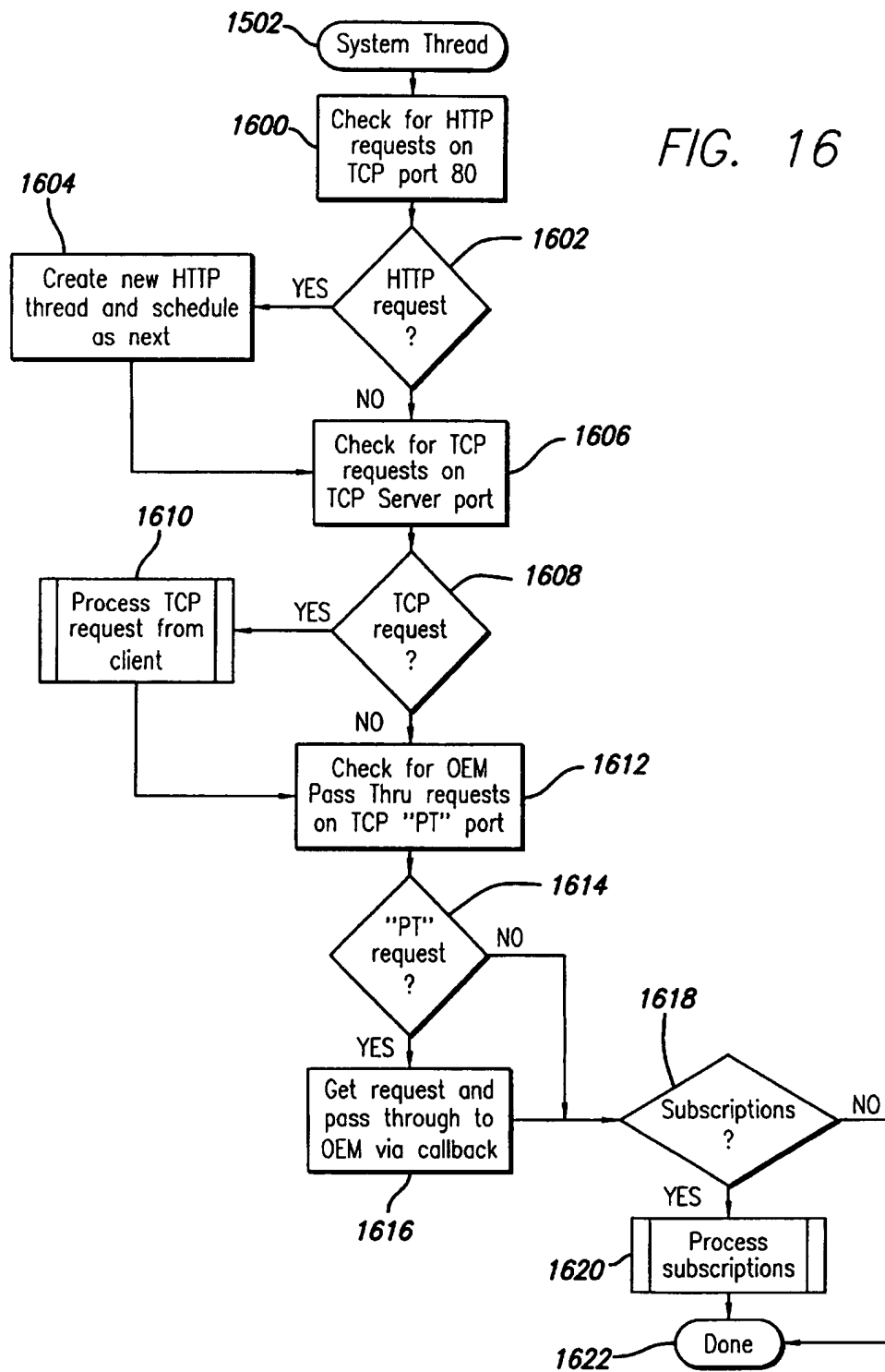


FIG. 15



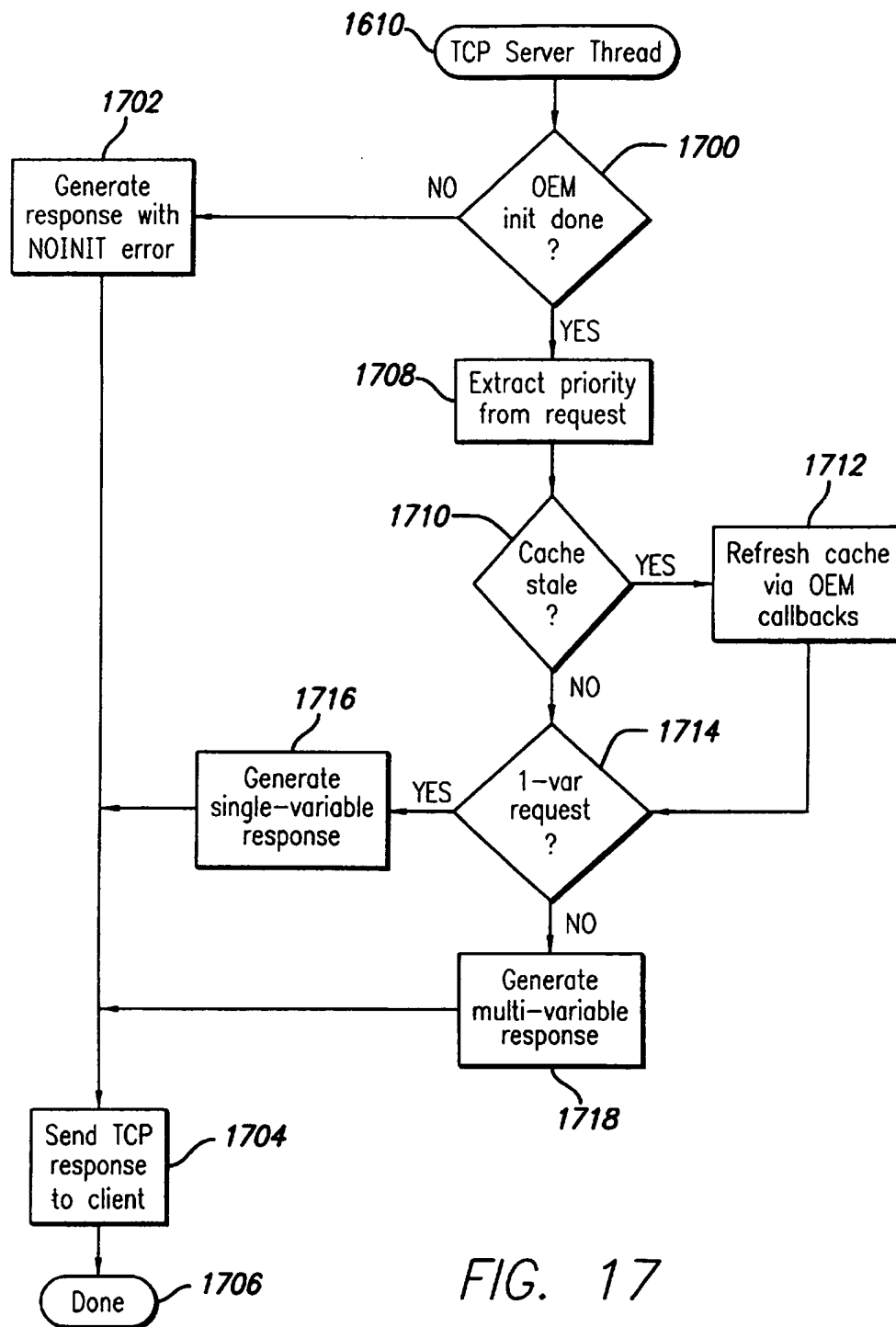


FIG. 17

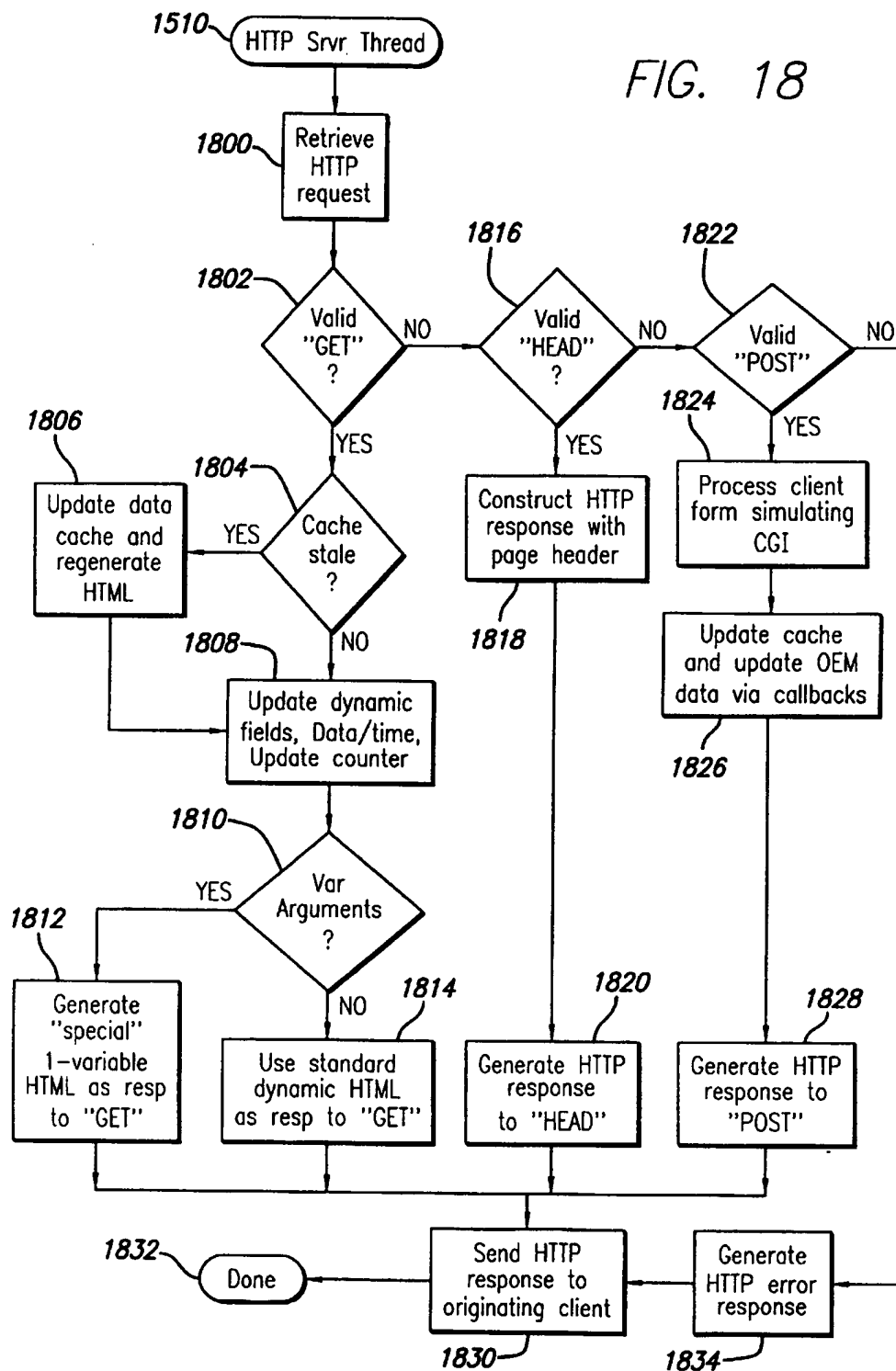
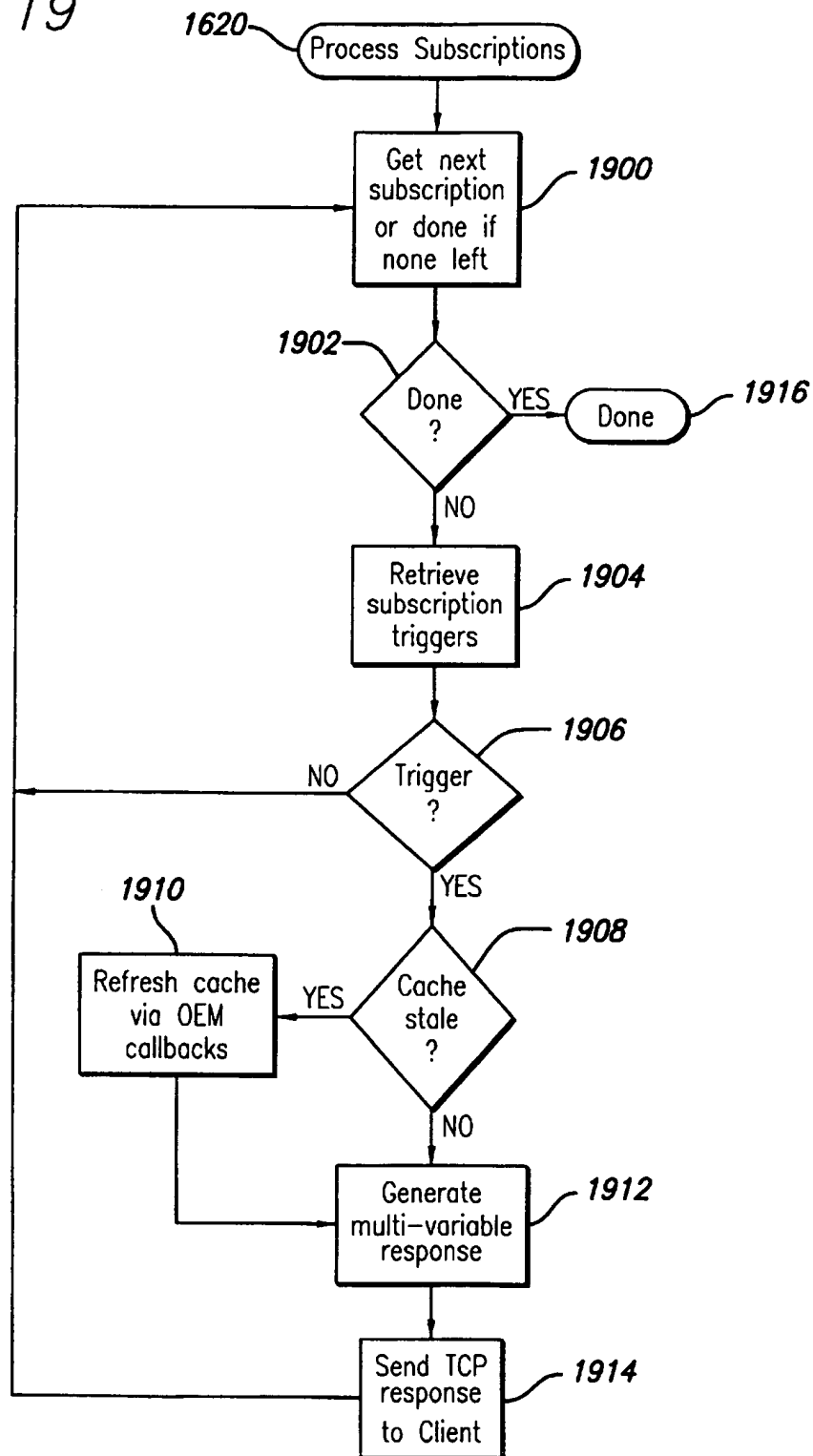


FIG. 19



SYSTEM AND METHOD FOR ACCESSING INFORMATION FROM A REMOTE DEVICE AND PROVIDING THE INFORMATION TO A CLIENT WORKSTATION

BACKGROUND OF THE INVENTION

1. Field of the Invention

This invention relates to providing information from a first device to a second device. More particularly, this invention relates to a micro-server system comprising a micro-server capable of being embedded into and/or associated with industrial equipment and/or consumer devices/products and capable of publishing information about such equipment and/or devices/products to thin clients running standard web browser software.

2. Statement of Related Art

Industrial process information is typically collected and used primarily through the offerings of a handful of key industrial information collection companies, or through internal home-brew solutions. Both approaches are costly to implement because the collecting architecture is very specific to the individual devices and to the whole process. Almost all the transport protocols are proprietary, and much of the media used to interconnect these devices, like RS-485, DeviceNet, and the like, are either proprietary or are limited to connecting only these kinds of devices. For a very remotely located device, options for connection can be severely limited.

In addition, most prior art solutions are hard-wired to the process, and a central host collects and manages device data, as is shown in FIG. 1. A central host is not a natural place for this information. The data is more timely, accurate, and meaningful at the device to which the data pertains.

Should the process change, re-wiring or re-instrumenting of prior art systems is typically needed. Re-programming the host is an enormous task fraught with the potential for bringing the whole process to a halt. The proprietary software that communicates with the host is usually licensed for, and installed on, each client computer, representing a big investment even from the update management perspective alone.

A significant drawback of such prior art systems is that if the host fails or is unreachable for any reason, all tactical and strategic data becomes unavailable.

Therefore, it is an object of this invention to simultaneously remove the host computer, as shown in FIG. 2 in which micro-server is abbreviated μ Server, remove the need for customer programming, unify the network fabric throughout factories and offices (Ethernet), provide secure access to any effector or device in the process from any workstation in the enterprise, and reduce the total cost of ownership.

It is an additional object of this invention to prevent information about a device from being maintained on a centralized host computer and to allow such information to reside in the actual device itself. With the information residing in the device itself, if the device is moved, its data moves with it. If the device is replaced, the new device can automatically publish its new data according to the principles of this invention.

It is a further object of this invention to enable devices to come on line and be browsable by a browser when the devices are shipped from the OEM. According to the principles of this invention, such devices could be capable of providing operational data, limits, suggested maintenance

cycles, specifications, links to the manufacturer's web site for detailed drawings, and literally whatever other information that the OEM desires.

Typically, original equipment manufacturer ("OEM") software professionals do not program with the Windows Application Programming Interface ("API"), and therefore almost never write code for a network. Typically, however, they are very familiar with the embedded software necessary to monitor and control industrial devices.

It is therefore a further object of this invention to abstract and encapsulate the highly complex TCP/IP network layer and Internet Web services to provide a simple, yet comprehensive, API in the "embedded" problem space, with which most OEM software professionals are familiar. It is a further object of this invention to publish information about the industrial equipment, also referred to as the device data, to an enterprise or the world as a web page on the corporate Intranet or the larger Internet.

SUMMARY OF THE INVENTION

A system for providing information about a first device to a second device. A micro-server interfaces with the first device to access the information from the first device. The information is then organized and formatted compatible with a communication protocol in preparation for making the information available to said second device. The information is made available to the second device while abstracting the communication protocol from the first device.

The system could include: an OEM application programming interface ("API") for interfacing between the first device's software layer and the micro-server; a TCP/IP stack for interfacing with a hardware interface; a TCP/IP API for providing access to the TCP/IP protocol stack; a hardware Ethernet interface; a system services API for providing the micro-server access to system services from the first device; an HTTP protocol server for satisfying interactive HTTP requests; a browser for interacting with the first device; a TCP/IP-based server for satisfying TCP/IP-based requests; a hyperlink to a website associated with the first device; a web-site associated with the first device; a default applet server for providing default applets to the second device to interface with the first device; a time server for providing current time information; an auto-discovery and view server for automatically detecting the first device being coupled to an interface; and a browser for interacting with the first device.

BRIEF DESCRIPTION OF THE DRAWINGS

FIG. 1 illustrates a prior art architecture including a centralized host computer for collecting industrial process information.

FIG. 2 illustrates a possible architecture, consistent with the principles of this invention, for publishing industrial process information.

FIG. 3 illustrates a simplified example system in accordance with the principles of this invention.

FIG. 4 is a simplified block diagram showing several subsystems of a micro-server according to this invention.

FIG. 5 illustrates several possible server/client types of combinations according to this invention.

FIG. 6 illustrates three primary communication paths between a micro-server-enabled device and client-side software.

FIG. 7 is a sample Read (Default) page.

FIG. 8 is a sample Control page.

3

FIG. 9 is part 1 of a sample Maintenance page, also referred to as a Maint page.

FIG. 10 is part 2 of the sample Maintenance page.

FIG. 11 is a sample Administration page, also referred to as a Admin page.

FIG. 12 is a partial sample of a Discover page.

FIG. 13 illustrates a three party model for online access to micro-server-enabled device documentation.

FIG. 14 is a simplified flowchart illustrating processing performed by a micro-server during micro-server initialization.

FIG. 15 is a simplified flowchart illustrating processing performed by a micro-server while getting run control from the device with which it is associated.

FIG. 16 is a simplified flowchart illustrating processing performed by a micro-server for executing a system thread.

FIG. 17 is a simplified flowchart illustrating TCP server thread processing by a micro-server.

FIG. 18 is a simplified flowchart illustrating HTTP server thread processing by a micro-server.

FIG. 19 is a simplified flowchart illustrating client subscription processing by a micro-server.

DETAILED DESCRIPTION

The operation of the micro-server is similar to that of a general-purpose web server. For example, FIG. 3 shows a remote thermostat device 300, labeled Temp Sensor in FIG. 3, equipped with a micro-server 302, labeled μ Server in FIG. 3, connected to the same Ethernet network 304 as a client workstation 306. The workstation could be used to monitor the remote temperature and/or to adjust the set point of the thermostat 300. The remote device contains two software components: (1) the OEM code 308 that performs the actual control functions and (2) the software of micro-server 302, which communicates with network clients. In this example, the workstation computer is considered a thin client, running no special software. All interactions with the remote micro-server-equipped thermostat device 300 is accomplished via a standard web browser program such as Netscape Communicator or Microsoft Internet Explorer. The micro-server-enabled thermostat device 300 could publish a standard web page containing the current temperature. Another page could be used to set the desired set point. Tremendous flexibility arises from the fact that multiple users, both local and far away, can access, view and change the information in the same manner without any special software, subject to configurable security measures.

Whenever a remote user accesses a micro-server-enabled device's web page, the micro-server software makes appropriate function calls to the OEM code to retrieve current information. Having done so, the micro-server incorporates the current information into the web page and sends the updated page to the requesting client machine. Similarly, whenever the user wishes to modify a control parameter, such as the set point of thermostat 300, a user could enter the desired temperature into a text box and submit the new information to micro-server 302. The micro-server 302 could then make appropriate function calls to the OEM code in order to change the setting.

In a first preferred embodiment of the invention, the micro-server is implemented in software. The micro-server could be provided to the OEM as a binary software library linked to the OEM application. Both the traditional static linking and the newer dynamic linking (DLL) methods are possible. In this embodiment, the OEM software could be

4

linked with the micro-server libraries, which would then become an integral part of the micro-server-enabled device's embedded software. The micro-server software could then be placed in the non-volatile medium in the OEM's hardware, such as EPROM or EEPROM.

The arrangement of characteristics listed in Table 1, below, could be used. As will be apparent to those skilled in the art, other suitable arrangements could also be used without departing from the scope of this invention.

TABLE 1

Characteristics of the First Preferred Embodiment	
Characteristic	Description
OEM software contained in	OEM hardware
micro-server software contained in	OEM hardware
TCP/IP stack contained in	OEM hardware
Ethernet Interface in	OEM hardware
OEM side of OEM-micro-server API	OEM code
micro-server side of OEM-micro-server API	micro-server code

In a second preferred embodiment, the micro-server could be an embedded-micro circuit board small enough to fit inside of the control box of a piece of industrial technology like a motor, a pump, a valve, or some other piece of process equipment. Of course, such an embedded-micro circuit board could also be placed into other types of devices and products, such as, consumer products. OEM software could be accommodated in the same board. In addition to the OEM and micro-server code, the software could include a TCP/IP protocol stack. As used in this specification and the appended claims, communication protocol dependent terms such as TCP/IP, HTTP, Ethernet, and the like, and means for supporting such protocols such as TCP/IP network protocol stack, and the like, are used for illustrative purposes only and should not be construed as limiting this invention to those particular protocols. As will be apparent to those skilled in the art, other appropriate communication protocols could also be used without departing from the scope of this invention. The hardware could include Ethernet support.

The characteristics of the second preferred embodiment of this invention could be arranged as in Table 2 below. As will be apparent to those skilled in the art, other suitable arrangements could also be used without departing from the scope of this invention.

TABLE 2

Characteristics of the Second Preferred Embodiment	
Characteristic	Description
OEM software contained in	hardware controller card
micro-server software contained in	hardware controller card

TABLE 2-continued

<u>Characteristics of the Second Preferred Embodiment</u>	
Characteristic	Description
TCP/IP stack contained in	hardware controller card
Ethernet Interface in	hardware controller card
OEM side of OEM-micro-server API	OEM code
micro-server side of OEM-micro-server API	micro-server code

In a third preferred embodiment, the micro-server could be a software component intended to run on a full WINDOWS platform (/95, /98, &/or /NT) or any other available operating system. This embodiment is useful in situations where the OEM's control platform is a computer capable of supporting a standard operating system environment. The operating system could provide TCP/IP support, and the computer's hardware could provide Ethernet connectivity. The characteristics of the third preferred embodiment of this invention could be arranged as in Table 3 below. As will be apparent to those skilled in the art, other suitable arrangements could also be used without departing from the scope of this invention.

TABLE 3

<u>Characteristics of the Third Preferred Embodiment</u>	
Characteristic	Description
OEM software contained in	OEM computer
Micro-server software contained in	OEM computer
TCP/IP stack contained in	OEM operating system or add-on
Ethernet Interface in	OEM computer
OEM side of OEM-micro-server API	OEM code
Micro-server side of OEM-micro-server API	micro-server code

In a fourth preferred embodiment, the invention comprises hardware that does not contain the OEM code, the OEM code being embedded in its own hardware. The micro-server software is contained in its own micro-controller, with an Ethernet interface and an external hardware interface (e.g., serial, parallel, PC-104, etc.) to connect the micro-server's micro-controller to the OEM's micro-controller. In this embodiment, the OEM-to-micro-server API abstracts the hardware interface between the OEM hardware and the micro-server hardware. This embodiment is particularly useful for retrofitting existing devices for web operation.

The characteristics of the fourth preferred embodiment of this invention could be arranged as in Table 4 below. As will be apparent to those skilled in the art, other suitable arrangements could also be used without departing from the scope of this invention.

TABLE 4

<u>Characteristics of the Fourth Preferred Embodiment</u>	
Characteristic	Description
OEM software contained in	OEM hardware
Micro-server software contained in	hardware controller card
TCP/IP stack contained in	hardware controller card
Ethernet Interface in	hardware controller card
OEM side of OEM-micro-server API	OEM code on OEM hardware
Micro-server side of OEM-Micro-server API	micro-server code on hardware controller card

A fifth preferred embodiment of this invention could use an embedded micro-controller built on a PC circuit board (ISA, PCI, etc.) that can be plugged into another computer. The micro-controller could contain the micro-server software and the TCP/IP network protocol stack. The OEM code could run on the computer that has the PC circuit board plugged into it.

TABLE 5

<u>Characteristics of Fifth Embodiment</u>	
Characteristic	Description
OEM software contained in	OEM computer
Micro-server software contained in	plug-in hardware card
TCP/IP stack contained in	plug-in hardware card
Ethernet Interface in	plug-in hardware card
OEM side of OEM-micro-server API	OEM code on OEM hardware
Micro-server side of OEM-micro-server API	micro-server code on plug-in hardware card

Each of the preferred embodiments comprises three Application Programming Interfaces ("APIs"): an OEM API, a TCP/IP stack API, and a System Services API. These APIs are suitable for use with current networking technology. It will be apparent to those skilled in the art that the concepts taught herein may be applied using API's other than those herein described and that the same or other API's may be used in connection with networking protocols developed in the future without departing from the scope of this invention.

The OEM API is used by the OEM software to configure the micro-server. The OEM API is also used to exchange data between the OEM software and micro-server software. The OEM API abstracts the micro-server to the OEM layer. The fundamental purpose of the OEM API is to provide a high level of abstraction for the micro-server from the OEM programmer's perspective. This allows the programmer to remain focused on the embedded application without having concern for the details of making the application web-enabled.

The OEM API is divided into five primary groups: (1) initialization group; (2) callback functions group; (3) system services group; (4) TCP/IP pass through group; and (5) scheduling group. An alphabetically organized listing of a micro-server OEM API functions including a synopsis of the call, the appropriate declaration, a narrative description of all arguments, the return value and associated usage notes are attached as Appendix A to this specification.

The OEM API Initialization group comprises a series of functions, which are called by the OEM software on power-up of the embedded device in order to inform the micro-server about variables which may be queried. For instance, a micro-server enabled servo-valve of the type typically

used in chemical processing could be capable of providing data on its current setting (0–100%), fluid flow rate, and temperature. To make this data accessible to the outside world, the OEM program could advertise it to a co-embedded micro-server by executing a separate call to an appropriate API initialization function. Each such call could provide the name of the variable, e.g., Flow Rate, the units of measure, and a pointer to a callback function to be used by the micro-server to retrieve the value of that variable.

The OEM program could also execute similar API functions to inform the micro-server about its control points. The example servo-valve could have a single control point: the desired setting (0–100%). It could therefore make an appropriate call via the API to provide information about this control point and to provide a callback function that could be used to adjust this setting.

The OEM API callback functions advertised to the micro-server by the OEM software, could be used by the micro-server to retrieve the values of the OEM variables or to change device settings. The micro-server calling these callback functions will typically make up the bulk of interaction between the OEM software and the micro-server. Significantly, the OEM software is typically essentially unaware that the micro-server is making these calls. This is consistent with the desired abstraction of the micro-server from the embedded OEM software.

Preferably, the micro-server software is completely portable. Therefore, it typically does not make any assumptions regarding the presence or functionality of an underlying operating system. In order to provide the micro-server with very rudimentary system services, for example, access to non-volatile storage or timer services, the OEM software provides these to the micro-server, again, via callback functions. This occurs during initialization. As is similar to the data and control callback functions discussed above, the execution of these system callback functions is transparent to the OEM program.

In a standard micro-server-enabled device, the OEM software would not generally talk directly to TCP/IP clients. Such communications are typically handled by the micro-server and are invisible to the OEM software. The OEM API does, however, provide the ability for the OEM software to open a TCP server port and service special requests directly. When such requests arrive on the specified port, the micro-server passes them through to the OEM software. This occurs via a callback mechanism similar to those discussed above. The responses from the OEM software to the originating client are passed through the microserver in the opposite direction.

The scheduling group of the OEM API is comprised primarily of a function that is used by the OEM software to run the microserver software after the microserver has been initialized. The OEM software or hardware typically would make arrangements to allow the micro-server to run periodically. Since most embedded applications generally do not have an underlying operating system, the execution of the micro-server is typically controlled by either calling it periodically via a function from the OEM API scheduling group. Such a function call could occur, for example, by explicit periodic calls from the OEM software or by attaching the function to a timer interrupt.

In addition to the OEM API, each preferred embodiment comprises a second API, the TCP/IP stack API, which is used by the micro-server to communicate with the underlying TCP/IP stack. The TCP/IP API could be standardized to conform to the Winsock 1.1 interface standard. As will be

apparent to those skilled in the art, other suitable interface standards may also be used without departing from the scope of this invention. The TCP/IP API abstracts the TCP/IP protocol stack to the micro-server.

Each preferred embodiment also comprises a third API, the System Services API, which is used to provide the limited service required by the micro-server. Since the micro-server is preferably software platform independent, the System Services API may be defined by the OEM code via the OEM API. As a consequence, the System Services API typically will not be usable by the micro-server until the initialization is complete. The System Services API provides the system services to the micro-server, while abstracting from the micro-server the details of providing such services.

The system services used by the micro-server during operation may be provided either by an underlying operating system or by the OEM code itself. The entity providing the services is typically transparent to the micro-server.

FIG. 4 is a simplified block diagram showing possible components of a micro-server. The OEM API is depicted by bold double-ended arrow 400. The System Services API is depicted by bold double-ended arrow 402. The TCP/IP API is depicted by bold double-ended arrow 404.

System services 406 provides the system services required by the micro-server. This software is abstracted from the micro-server's point of view and may be either a part of the native operating system or part of the OEM software layer. Embedded OEM Software layer 408 communicates with device-specific hardware, except for the Ethernet hardware interface. Device hardware 410 may include sensors and other inputs, effectors or other outputs, and the like. The Ethernet hardware interface could be included in this layer, but is preferably accessible only from the TCP/IP stack. The main body 412 of the micro-server could comprise: micro-server-OEM API 414; micro-server-TCP/IP API 416; synthesized HTML 418; OEM data image 420; HTTP server 422; and TCP/IP based server 424.

Micro-server-OEM API 414 is an interface that could be used primarily by the OEM layer 408 to configure the micro-server and by the micro-server to exchange data with the OEM layer. The TCP/IP protocol stack could be used for communications over an Ethernet. TCP/IP protocol stack as used in this specification and the appended claims is not intended to be limited by the TCP/IP protocol. Rather, TCP/IP protocol is merely illustrative, and not intended to limit the scope of this invention to TCP/IP protocol. Other suitable protocols could be used without departing from the scope of this invention. Micro-server-TCP/IP API 416 is used by the micro-server to access the TCP/IP protocol stack 426.

Synthesized HTML 418 can contain a dynamic copy of the HTML version of the web pages served by HTTP server 422 to a client. OEM data image 420 can cache the OEM layer data in order to minimize callback requests from the micro-server. Stale OEM data can be automatically refreshed. HTTP protocol server 422, also referred to as HTTP server, can be used by the micro-server satisfy interactive HTTP request typically from thin browser-based clients. TCP/IP Based Server 424 could be used by the micro-server to satisfy TCP/IP based requests from clients executing default applets, manufacturer-supplied applets, client-side API software, or other TCP/IP-based software.

Each of the preferred embodiments is capable of five primary operating modes, as shown in the following table:

TABLE 6

Micro-Server Operating Modes		
Mode	Operating Mode	Client Software
1	HTTP Server	Standard interactive browser.
2	HTTP Server with Client-side Refresh	Standard interactive browser
3	TCP/IP Server	Default applet, OEM-provided applet, Client-side API, Custom client
4	TCP/IP Server with Server-side Push	Default applet, OEM-provided applet, Client-side API, Custom client
5	TCP/IP Pass-through	OEM-provided applet

FIG. 5 depicts several possible server/client types of combinations according to the principles of this invention. The five micro-server operating modes are depicted by the boxes on the left side of FIG. 5. They are: HTTP server 500; HTTP server with client-side refresh 502; TCP/IP Server 504; TCP/IP Server with server-side push 506; and TCP/IP pass through mode OEM server 508. The five operating modes function as follows.

HTTP server 500 is the most common mode used in interactive access to a micro-server-equipped device from a thin-client workstation running a web browser. When a user lands on a desired device's default page, an appropriate HTML description of the current state of the device is dynamically configured by the micro-server and the resulting page is displayed on the client work station's screen. This information is typically not updated dynamically, and the user typically would click on the browser's RELOAD button to cause an update to occur. As depicted in FIG. 5, HTTP server typically operates in conjunction with a standard browser 510. A standard browser could include, but is not limited to, Netscape Communicator or Microsoft Internet Explorer. Either of these products, as well as many others both commercially available and custom designed, are capable of providing a sufficiently interactive client interface to access micro-server pages.

HTTP server with a client-side refresh 502 is similar to the HTTP server mode 500 except that the page displayed in the browser is continually updated at fixed time intervals. HTTP server with client-side refresh will typically be available if the micro-server-equipped device has been appropriately configured by the user. This mode also typically depends on a client-side refresh configured into the HTML description of the page being viewed. HTTP server with client-side refresh 502 typically operates in conjunction with a standard browser 510, as shown in FIG. 5.

The TCP/IP server mode 504 is non-interactive. Data is exchanged between a client program on the remote workstation and a TCP/IP server within the micro server. As shown in FIG. 5, client applications which typically use this mode include: (1) a default client-side micro-server applet which determines the micro-server-equipped device's configuration and configures itself 512; (2) an OEM-provided applet which adheres to the standard micro-server TCP/IP application packet format 514; or (3) a micro-server client-side API application 516.

A default applet 512 is a software entity provided by an applet server. Once acquired by a client, it becomes a part of the client browser and it will typically be capable of displaying device data in a convenient, graphical manner. The default applet could be written according to a standard micro-server application-to-application protocol definition

and could communicate directly with the TCP/IP server within a micro-server. Since this communication takes place within the micro-server, underneath the OEM API, which provides networking abstraction, it is typically invisible to the OEM layer.

An OEM-provided applet 514 is functionally equivalent to the default applet, but may provide a richer set of functions. It could be acquired from the OEM either directly or via an applet procurement agent. The OEM-provided applet could communicate with the micro-server in different modes. If written to the standard micro-server application-to-application protocol definition, it communicates directly with the TCP/IP server within the micro-server, just like a default applet discussed above. The OEM could also write the applet to use a different application-to-application protocol. In this case, the applet would communicate with the OEM layer via the TCP/IP pass through mode 508. This method would typically be discouraged because, under such circumstances, the micro-server would no longer provide full networking abstraction to the OEM software.

The Client-side API 516 is a software entity that can execute on the client machine and provide programmatic access to micro-server functions via an API. The Client-side API could use the TCP/IP server within the micro-server. In addition to handling ad-hoc requests from client applications, it is sufficiently intelligent to listen for broadcast messages from micro-servers to which it subscribes.

TCP/IP server with server-side push 506 is a mode similar to the standard TCP/IP server mode 504 but includes a limited push or broadcast capability to automatically initiate update data transfers to a limited number of selected clients. Data transfer destinations as well as trigger issues that initiate them may be user configurable. Judicious use of this option can result in significant savings in network traffic. As shown in FIG. 5, TCP/IP server with server-side push typically operates with the same types of client-side entities as the standard TCP/IP server mode.

TCP/IP pass through 508 is a mode intended for special cases outside of the scope of normal micro-server operation. A specific example of the use of this mode is a scenario in which an OEM supplied applet 518 is running under the client-side browser, communicating with a specialized TCP/IP server in the OEM application itself. In this scenario, the micro-server simply passes all received TCP/IP packets to the OEM application via a prearranged call back function and replies to the originating client with packets provided by the OEM software. This functionality is supported by the micro-server for completeness and is typically not preferred because it does not take advantage of the network abstraction provided by this invention. Similarly, custom applications could be written that would execute on the client and make use of the TCP/IP server within the micro-server. Although such a use of TCP/IP pass-through mode is possible, it is not preferred because it does not take advantage of the network abstraction provided by this invention.

FIG. 6 illustrates that communications between the micro-server-enabled device and client-side software typically occur over three paths: (1) HTTP-protocol based communications between the HTTP server and the browser client, as shown at 600; (2) TCP/IP based communications between client-side default or OEM-supplied applets and the TCP/IP based server, as shown at 602; or (3) TCP/IP pass-through communications between the special client-side applications and the OEM software, with the micro-server acting as a pass-through intermediary, as shown at 604. While paths 600, 602, and 604 could all be used simultaneously by one

or more clients, typically path 600 will be used most often, while paths 602 and 603 will be used less often than path 600.

A micro-server-enabled device could have five built-in web pages. These pages could all exist at the top level of the micro-server web site, so that accessing each would be easy. If no opening page is specified when a web server is browsed to, the web server opens a page called default.htm, also referred to as the site's home page. Another way to access the home page could be by adding/read to the end of the URL. The home page for a micro-server-enabled device could publish the device's parametric data.

The control page is available by adding/control to the end of the URL. This page provides authorized persons access to the control functions, if any exist, on the device. For example, a servo-valve could be outfitted with the ability to set its flow rate from 0 to 100%.

The maintenance page is available by adding/maint to the end of the URL. This page could provide access to the maintenance functions and/or the maintenance record of the device.

The administration page is available by adding/admin to the end of the URL. This page is used by authorized persons to set operating characteristics of the micro-server such as the physical location name, the out-of-limits parameters for voltage, current, flows, and temperatures, push IP addresses, and the like. In general, it is used to tailor the device to its environment.

The discover page can be accessed by adding/discover to end of the URL. This page supplies information to clients during automatic discovery mode.

In order to clarify how the micro-server pages are accessed, assume that a unit has been configured with an IP address of 202.133.3.4. In addition, a local DNS server has mapped the same IP address to a URL of ServoValve37. Table 7 below demonstrates how the four pages are accessed either with or without the DNS server:

TABLE 7

Accessing micro-server Pages with or without DNS		
Micro-server Page	With DNS	Without DNS
Home (Default)	http://ServoValve37	202.133.3.4/default
Control	http://ServoValve37/control	202.133.3.4/control
Discover	http://ServoValve37/discover	202.133.3.4/discover
Administer	http://ServoValve37/admin	202.133.3.4/admin
Maintenance	http://ServoValve37/maint	202.133.3.4/maint

In order to simplify navigation, micro-server could automatically provide each page with links to the other four pages. FIG. 7 shows a typical micro-server page, in this case, a default page for a device controlling a Servo-valve. The screen is divided into two frames, the upper reflecting mostly static information and the lower containing the device read data. Buttons located on the left-hand side in the upper frame lead to the other four micro-server pages. As can be seen, the micro-server presentation can be very much like that of a standard page on the world-wide-web. This choice of presentation allows the use of client-side software, in this case a standard Netscape browser, which is familiar to most people. As will be apparent to those skilled in the art, the web pages presented in FIGS. 7-12 are representative of one possible implementation and other suitable implementations are also possible without departing from the scope of this invention.

In FIG. 7, the Auto Refresh feature is depicted as turned off in the upper right hand corner of the sample control page. Therefore, the page, as displayed in the browser, would remain static even though the device data may have changed. If the Auto Refresh feature were on, the page would be updated periodically. Underneath the Auto Refresh indication, the page contains both the current micro-server device time as well as the time of the last update. The latter refers to the time when the last set of data readings was obtained by the micro-server from the OEM layer. All data values are typically updated simultaneously.

The following fields shown in FIG. 7 would typically have been configured by the administrator, either via the Admin page or via the Client-side API: User Description; User Id; User Location; and Nominal, Low, and High Data Values for all device parameters. Once configured, these fields not only personalize the device but also change the way the micro-server reports data.

All device parameters are tagged with a unique identifier as depicted in the Id. column of the home page depicted in FIG. 7. Although not very important in the interactive presentation depicted in FIG. 7, this is a significant feature of the micro-server according to this invention. It allows parameter level access to micro-server data via the Client-side API. In addition, and also of significance, is the ability to access individual device parameters in other HTML constructs, allowing construction of alternate and/or hierarchical device and system views.

FIG. 8 depicts a sample control page. Any modifications of operating parameters could be subject to a security restriction imposed by a system administrator.

FIGS. 9 and 10 depict parts 1 and 2, respectively, of a sample maintenance page. The maintenance page could provide a very useful maintenance log that could be stored in non-volatile memory. If non-volatile memory is not available in the micro-server-enabled device, the maintenance log could be kept in a local maintenance server. The manufacturer's micro-server license Id. and the device's serial number, both of which could be available as micro-server variables, could form a unique identifier to be used in tracking the maintenance functions of the device.

FIG. 11 depicts a sample Admin page. FIG. 12 depicts a sample Discover page. The contents of the Discover page can be used by any authorized client to establish detailed understanding of a newly discovered micro-server-enabled device, for instance, a micro-server enabled device recently coupled to a network. Typically, micro-server variables uniquely identify important pieces of information related to the characteristics and the operation of the micro-server-enabled device. Micro-server variables may be used by the application-to-application protocol supported by the TCP/IP based server within the micro-server. Micro-server variables would typically be involved with the operation of the client-side API.

Web pages are expressed in HTML (HyperText Modeling Language) before they can be displayed by a browser. When a client browser lands on a given page, it requests the HTML description of the page to be sent. The HTTP server responds by sending the HTML and the browser, having received it, displays it on the client computer's screen. Upon receiving a request for HTML from a client, the micro-server makes appropriate calls to the OEM layer in order to retrieve appropriate data, constructs an HTML description of the requested page, and then sends the page to the client.

To reduce the micro-server's overhead of frequent calls to the OEM layer for retrieving data updates, the micro-server

13

can cache OEM data. The cache may include a date/time stamp of the last update. Upon receiving an HTTP request from a browser client, the micro-server can check whether the data is available in the cache (it usually is, except for during startup) and then checks the cache timestamp. If the data is reasonably recent, no update is performed and the last version of the dynamic HTML is sent. If the data is stale, however, the micro-server can update it by performing the required callbacks to the OEM layer, update the cache and the associated timestamp, re-construct the HTML, and send it to the requesting client.

The maximum data latency for HTTP clients can be specified via the OEM API by calling a function defined by the OEM API. The maximum data latency is the maximum acceptable age of the data cached by the micro-server, as served to HTTP clients. It would typically be the responsibility of the OEM to make reasonable estimates for this parameter in order to reduce the overhead of frequent updates and re-construction of the HTML page description. For example, if the parameter measured by the micro-server-equipped device is temperature and the sensor has a limited frequency response, specifying a short latency time would be counterproductive because frequent data updates would not be necessary and would consume excessive resources.

Maximum data latency for TCP/IP clients is controlled via a different OEM API function call. This parameter is typically smaller than the maximum data latency for HTTP clients since TCP/IP clients are usually higher in performance and not subject to the delays of the human-machine interface such as in an interactive HTTP-based browser.

From the client perspective, a micro-server appears to be a normal, fully functional HTTP server executing on a conventional computer with a fully functional file system. The reality, however, could be quite different. A typical hardware platform could comprise an embedded processor without a real file system. Furthermore, the amount of non-volatile memory available to a micro-server could be quite small. In order to maintain full HTTP compliance, the micro-server, typically abstracts or hides a number of important details from the client, maintaining full browser compatibility.

In the preferred embodiments, by default, the micro-server will not broadcast any data unless the device has at least one registered subscriber and at least one trigger condition has been specified.

The configuration of server-side push could be performed interactively via the browser interface and the Admin page or could be performed via the TCP/IP application-to-application protocol. In either case, it could be subject to security checks. A subscribing client may choose to unsubscribe from broadcasts from any micro-server-equipped device by removing its host name or IP address from the subscription list. Such un-subscription requests could be subject to normal security measures. The broadcast data could be sent to the IP address of the subscribing host or hosts and a specified port, such as TCP/IP port 162. Client-side software entities which have registered their subscription with any micro-server-equipped devices would typically have to be listening to this port in order to receive the broadcasts.

In the pass-through TCP/IP mode, an OEM-supplied applet could be running under the client-side browser, communicating with a specialized TCP/IP server in the OEM application itself. In this scenario, the micro-server could pass all received TCP/IP packets to the OEM application via a prearranged callback function and could reply to the originating client with packets provided by the OEM software.

14

The OEM layer could use the OEM API to request the micro-server to set up multiple pass-through connections of this type, each with an arbitrary associated TCP/IP-port number. The choice of the port number should not conflict with standard port numbers used by the micro-server or any other well known port numbers as defined in standard TCP/IP RFCs. The responsibility to use correct port numbers would typically lie with the OEM layer.

The micro-server is typically initialized each time power is applied to the device into which it is embedded or to which it is associated. The sequence of events could proceed as follows: (1) power is applied to the device; (2) the OEM layer is bootstrapped; (3) the OEM layer performs various initializations (other than initialization of the micro-server); (4) the OEM layer retrieves the IP address of the device from appropriate hardware such as a dip switch, or the like; (5) the OEM layer begins the micro-server initialization; (6) the OEM layer performs the micro-server initialization via the OEM API, passing the micro-server the IP address; (7) the OEM layer terminates micro-server initialization; (8) the OEM layer makes arrangements to start the micro-server execution thread; and (9) the OEM layer initiates its normal execution. With respect to step (4), alternately, a dynamic IP address assignment could be obtained from a local DHCP server.

The micro-server is typically not operational until the initialization has completed and an execution thread has been arranged for it, at which point, it typically starts operating normally.

Auto-discovery enables new micro-server-enabled devices to become useful immediately after plugging them into a network. Auto-discovery typically occurs in two separate steps: (1) new devices are automatically discovered by an auto-discovery and view server that scans the network for new devices. The scans can be configured to occur periodically or can be run on-demand. Once discovered, a new device can be automatically inserted into the main view on the server and can be accessed simply by clicking on its name. This operation could cause the client browser to be transferred to the device's home page; and (2) an automatic attempt can be made to load an OEM provided viewing/control applet to the client computer. If this operation succeeds, the applet can become operational immediately. If the attempt fails, however, a default applet can be loaded from the Local Applet Server. The applet can be initiated, causing it to perform the second, detailed discovery of the node itself. The TCP/IP application-to-application protocol is used to glean from the node, information about its characteristics, variables, recommended settings, etc. The same information could be presented in human-readable form on the device's discover page.

An OEM could configure the micro-server to include standard HTML links to relevant device documentation located on an OEM web site or some other web site associated with the device. This configuration of the micro-server could take the form of a simple call to the micro-server via the OEM API.

This call could result in a Documentation link created on the device's Maintenance page. Immediate and seamless access to on-line documentation, including drawings, specifications, parts lists, and manuals, of any micro-server-equipped device could be provided simply by clicking on the Documentation link on the device's maintenance page from any client browser. FIG. 13 illustrates how this could work.

In the preferred embodiments, access to online device documentation located on the OEM's web site could be a

four-step process, as illustrated in FIG. 13. First, the user could click on a Documentation link created on the device's Maintenance page, as indicated by arrow 1300. Second, the URL of the documentation could be sent back to the client browser by the HTTP server component of the microserver; as shown by arrow 1302. Third, the browser could request the documentation, as specified by the URL, from the OEM's website, as depicted by arrow 1304. Fourth, the browser could load the requested documentation from the manufacturer's site as shown by arrow 1306. Advantageously, although the online documentation exists on the manufacturer's web site, the hyperlink provided on the micro-server maintenance page makes it appear that the documentation actually resides in the device itself.

The operation of the micro-server could be language independent. Object files containing the fixed string content for various languages could be provided. Parameter description strings, maintenance log entry strings, etc., as passed to the micro-server via the OEM interface, could be language independent. In addition, a country code could be supported for configuring certain internationalization features, including, but not limited to, regional seasonal time shifts. In an appropriately configured micro-server, simultaneous multilingual operation is possible. Similarly, a micro-server could be configured to report time to each client in each client's local time.

Time could be maintained as Universal Coordinated Time (UTC). The time information could be obtained via an appropriate API interface advertised to the micro-server via the OEM interface. Although the notion of UTC may not necessarily be needed in a given application, it could be extremely useful when coordinating real time information from multiple sources.

If an IP address or a hostname of a local time server has been specified, the micro-server could attempt to obtain local time information directly and adjust its clock accordingly. If a local time server is not available and the country code has not been set, the micro-server could be configured with the values of the time difference from UTC and any local/seasonal time offset. Regardless of how the date and time are originally set, they could continue to be available as both UTC and local values.

The micro-server's notion of time could be based on local time obtained from system services and corrected with a local offset in hours from UTC contained in a variable TZ1. Application of the offset could provide the local time. The notion of daylight-savings time as well as any special regionally-based time offsets could be handled by a time internationalization function within the micro-server, resulting in an appropriate value of a variable, TZ2. This function could be used in the event that a timeserver could not be accessed on the subnet. The local time could be obtained by summing the UTC with TZ1 and TZ2, as follows:

$$T_{local} = T_{UTC} + TZ1 + TZ2$$

If a local time server is not available and the country code has not been set, the micro-server could be configured with the values of the time difference from UTC and any local/seasonal time offset (variables TZ1 and TZ2).

The micro-server could use hostnames, IP addresses, and URLs, in different contexts. IP addresses are, effectively, hardware addresses of computers communicating over Ethernet. The micro-server could use IP addresses to specify addresses of clients subscribing to automatic data updates (Server-side push), the address of a local Timer Server, and the like.

In all cases where a machine has a specific name, for example, Bldg45_host, that name could be used in config-

uring the micro-server. This approach could provide added flexibility, since the IP addresses of a given machine can be changed without the need to re-configure micro-server-equipped devices which reference it. To accomplish this, the micro-server could make use of a DNS (Domain Name Service) in order to look-up the IP address of a host. This service could be provided by a DNS server connected to the same network as multiple micro-servers. In order to use the DNS service, micro-servers could be configured with an IP address of a local DNS server.

URLs, or Uniform Resource Locators, are typically addresses of a specific file or another Internet resource. They are typically formed by adding a file pathname to the address of the device. The hostname or an IP address can be used as a base-name for the path, for example VTG_applets/servoalves/GWLinc/model3500, or 124.67.98.1/servoalves/GWLinc/model3500. URLs could be used by a micro-server to point to items such as manufacturer documentation for a specific device, an Applet used to view the device's data, and the like.

A micro-server could support Parameter level URL (PURL). A PURL can be used to reference a particular parameter published by a micro-server-equipped device. For example, the PURL slurrypump3/da0013.dat,value could refer to the value of data parameter 13 on micro-server called slurrypump3. Similarly, 128.45.33.11/mfgname.dat could return a string containing the manufacturer name of the micro-server-equipped device at IP address 128.45.33.11. The qualifier which follows the comma (,) separator could be an additional part of a request to specify an attribute of the parameter to be retrieved.

The operation of micro-server-enabled devices could be augmented by several additional software components such as: Default Applet Server; Time Server; Maintenance Server; Auto-discovery and View Server; Applet Procurement Server; and Local Applet Server/Procurement Agent.

While applets are not necessary for accessing micro-server-equipped devices with a browser, the Default Applet Server could provide default applets for interfacing with micro-server devices from within a standard client browser. The default applets could be used in situations where either (1) the manufacturer did not provide a specific viewing/control applet, or (2) the applet cannot be reached because of insufficient network connectivity. The server could contain standard default applets and distribute them, if necessary, to the browser clients. This operation could be handled by an applet procurement script embedded into the HTML code loaded from the micro-server-enabled devices.

The Time Server could provide the current UTC time to the requesting micro-server nodes, upon request. In addition to the UTC time, it could provide the hour offset from UTC and a second local time offset. Both offsets could be in hours. The Time Server could present a standard micro-server interface, in which case its services could be accessed by any software client via the standard client-side API.

The Maintenance Server could be a specialized entity used to provide maintenance logs for those micro-server-enabled devices that have a very limited amount of, or no, non-volatile storage. The server could essentially form an external repository for maintenance data hyper-linked to the device, so that the maintenance server's operation would be essentially transparent to the client.

The Auto-discovery and View Server could be a focal point of user access to a web of micro-server-enabled devices. The Auto-discovery and View Server could automatically discover micro-server devices within a given range of IP addresses. It could be run on demand or

configured to run periodically to update the mappings. Auto-discovery could allow new micro-server enabled devices to publish device data simply by plugging the device into a network. The Auto-discovery and View Server could generate a local HTML master index page (Main View) of all devices, usable by a browser. This list could become a focal point for many users. Individual access points (URLs) could, of course, be book-marked on a user's browser in a manner well known in the art.

The Auto-discovery and View Server could generate a local database file (text, Access, etc.) with all device mappings, useable by other programs (applets, applications, etc.). This list could be useful for configuring devices remotely.

The Auto-Discovery and View Server could allow the system administrator to configure system-wide monitoring and control panels that could present different views of any number of devices in the system. Each data point, or any other micro-server variable, could be monitored individually by referring to the variable's PURL. Once configured, views could be served-up to other clients and the data could be monitored live via the established views. Typically, an OEM applet would be available either locally or from the OEM server and would interact with the Auto-discovery and View Server.

The Local Applet Server could locate and retrieve device-specific applets from an OEM's server and make them available on a local facility server for access by clients. If direct Internet access is limited due to restricted connectivity, the Local Applet Server could attempt to procure the correct applets from the manufacturer via electronic mail. Once the applets are procured, they could be cached on the Local Applet Server. If the applets cannot be procured, and before they can be successfully procured, the Local Applet Server could use a substitute default applet. Once loaded on the client machine, the default applet could perform detailed discovery of the device. References to these client applets could occur on the micro-server pages that are served up. Since the applets are usually cached on the client machines, the overhead required to load them would typically be experienced only on initial use.

There are several types of client software which could interact with the micro-server. They are Standard Browser; Client-side API; Standard Applet; OEM-provided applet; and Custom Software.

Any standard browser (Netscape, Internet Explorer, etc.) running on any workstation platform (WINDOWS) 95/NT, UNIX, etc.) can be used to interact with micro-server-enabled devices or any of the servers listed above. OEM-provided viewing/control applets could be automatically loaded if desired. The location of a thin browser is essentially irrelevant. It could be connected to the same sub-net as the devices or it could access the network remotely. The workstation used to interact with a device can be chosen ad-hoc, since no special software is needed.

The purpose of the client-side API is to provide program access to the micro-server-enabled devices and other servers. It could allow custom monitoring and control applications to be developed and executed on the client, with full access to the device and server data. The client-side API could provide multi-threaded access to micro-server-enabled devices and servers. It could be used by standard OEM-provided viewing/control applets as well as by the default applets. The client-side API calls could allow reading and writing of individual variables from/to a specified device, subject to configurable security restrictions, and it could perform auto-discovery of devices that it communi-

cates with. The client-side API code could perform local caching of device data. This would reduce network traffic since some of the program data requests, depending on requested priority, could be satisfied locally.

OEM-provided applets could be interactive viewing/control software modules that could be automatically loaded into the client browser from the OEM web site. This operation can be totally transparent to the client.

OEM-provided applets could be matched to the micro-server-enabled eliminating any need to perform detailed discovery. Such applets could check and validate the device OEM release level and disable themselves if they were no longer current. The applet procurement mechanism could then be re-activated, repeating the above cycle.

Default applets could be general purpose in nature, perhaps lacking some of the features of OEM applets. They would typically be uniform in appearance and devoid of the personalization of OEM applets. Default applets could be obtained from the local applet server.

Custom software developed for the client could use the client-side API to communicate with micro-server-enabled devices or servers.

The micro-server software is comprised of multiple tasks, referred to as threads. The individual threads are preferably maintained by the micro-server software itself. The micro-server software typically does not block, since that would likely interfere with operation of the OEM code which is responsible for operation of the micro-server-equipped device itself. The micro-server software typically gains processor control periodically and preferably relinquishes such control as soon as is reasonably possible.

The micro-server could run only when it is called at one of its standard entry points. The initialization API calls could account for the majority of these and could be used to configure the micro-server software based upon the device in which the micro-server software is embedded or with which it is associated. The OEM code could make these calls upon each bootstrap.

Referring to FIG. 14, processing is depicted as being initiated by the OEM software layer at 1400. If the OEM software layer has called `us_InitStart()`, the software begins the initialization process, as shown at 1402, 1404, and 1420. If `us_InitStart()` has not been called, any other initialization function calls are simply ignored, as indicated at 1406, 1408, and 1410.

The OEM code calls `us_InitDone()` to signal the end of initialization, triggering validation of the initialization sequence, as shown at either 1412, 1414, 1416, and 1410 or 1412, 1414, 1416, 1418 and 1420. When the OEM layer calls `us_InitDone()`, the micro-server could validate the entire initialization sequence as shown at 1414. If the initialization sequence was successful, the TCP/IP protocol stack can be initialized to enable communications, as shown at 1416 and 1418. The TCP/IP stack could be responsible for the initialization of the underlying TCP/IP hardware, the detailed workings of which are abstracted from the OEM layer by the micro-server. A call to `us_InitStart()` could clear all of the micro-server data structures and could be the first step of the initialization process. Any OEM call with inappropriate arguments could cause a failure.

If the call was neither `us_InitStart()` nor `us_InitDone()`, then it could be processed as either an initialization call or a TCP passthrough request being made by the OEM code, as shown at 1406, 1412, and 1422. TCP passthrough requests could cause the micro-server to blindly pass TCP/IP requests through. The validity of the call could be checked. Arguments and context could be validated for both initialization

and TCP passthrough calls. If all is well, the return code could inform the OEM accordingly, as shown at 1424 and 1420. Similarly, any errors could result in an appropriate error code being returned, as shown at 1424, 1408, and 1410.

Referring to FIG. 15, the micro-server could gain control via an explicit call from external software, as shown at 1500. This could occur via an explicit call to the function `us_Run()` made periodically from the OEM software or from an interrupt service routine tied to a timer. This is the principal entry point for post-initialization processing by the micro-server. Once entered, the micro-server could execute a system thread, as depicted at 1502, and one thread associated with an active HTTP connection, if any exist. The details of the system thread are presented in FIG. 16. The entire execution point is preferably non-blocking in order to return quickly to the mainstream OEM software.

If there are no active HTTP connections, processor control can be returned to the OEM software, as depicted at 1504 and 1506. If there are active HTTP connections, the next HTTP connection thread to be executed can be selected from a process table, as depicted at 1508. Since there is at least one active HTTP connection thread, an HTTP connection thread can be executed, as depicted at 1510. The details of executing an HTTP Connection thread are presented in FIG. 18.

If the HTTP connection thread execution has completed, the HTTP connection thread can be removed from the process table and processor control can be returned to the OEM software, as depicted at 1512, 1514, and 1506. If the HTTP connection thread has not yet completed, for instance, because it is waiting for an external event, the HTTP's connection thread execution state can be retained in the process table.

Referring to FIG. 16, the System thread can be entered each time the micro-server software executes after initialization has been successfully completed. The system thread could perform all micro-server functions except handling HTTP connections. The System Thread could initially check for any HTTP requests on the TCP socket bound to port 80. Port 80 is the standard port for receiving HTTP requests originated by client browsers. If there is a pending HTTP request, a new HTTP connection thread could be created and scheduled as the next HTTP connection thread to be executed, as depicted at 1600, 1602, and 1604. New HTTP connections are preferably scheduled as high priority since accepting the HTTP requests as soon as they arrive allows the micro-server to process them more quickly.

Requests on the TCP server port originate from client entities other than standard interactive browsers and could be handled differently, according to a traditional client-server model, as depicted at 1606, 1608, and 1610. Such requests typically originate from client side applets or micro-server APIs. If there is a pending TCP client request, the TCP request could be processed and a response is sent to the originating client. Details of this process are presented in FIG. 17.

If there is a pending TCP pass-through ("PT") request on the ports specified by the OEM software during initialization, the TCP pass-through request is processed and passed to the OEM software via a callback function specified during initialization, as depicted at 1612, 1614, and 1616. Pass-through requests are literally passed through by the micro-server, in both directions, without interpretation.

If there are any subscribers, client subscriptions could be processed, as depicted at 1618 and 1620. Details of subscription processing are presented in FIG. 19. Subscribers are clients which have registered with the micro-server for

automatic push notification upon the existence of certain triggers. As depicted at 1622, processor control is then returned to the Get Run Control thread depicted in FIG. 15.

Referring to FIG. 17, the TCP server thread can process requests for data originating from non-browser clients. These clients typically are client-side applets or micro-server APIs. This thread can be entered from the System thread upon detection of the presence of such requests.

Initially, the TCP Server Thread could check whether OEM initialization of the micro-server has been successfully completed, as depicted at 1700. If not, as depicted at 1702, 1704, and 1706, an error packet could be returned to the originating client and control could be returned to the System Thread depicted in FIG. 16. If the micro-server has been successfully completed, the priority could be extracted from the client request, as depicted at 1708. The priority could be used to determine whether the data in the OEM data cache is sufficiently current to satisfy the request. The concept of request priorities and OEM data caching could be used to satisfy client requests with reasonably recent OEM data without having to get those data values from the OEM upon receiving each request. If the OEM data is stale (i.e., not sufficiently current), the cache could be refreshed by obtaining current OEM data via execution of callback functions specified during the initialization process, as depicted at 1710 and 1712. The cache could be time-stamped each time this happens.

If the client is requesting the value of a single micro-server variable, a single-variable response packet could be generated, as depicted at 1714 and 1716. If not, a multi-variable response packet could be generated, as depicted at 1718. As depicted at 1704, and 1706, the response packet could be sent to the originating client and control could be returned to the micro-server System Thread as depicted in FIG. 16.

Referring to FIG. 18, the HTTP server thread could be executed for each incoming HTTP request originated by a client browser or another HTTP-compliant entity. The HTTP server thread could be called from the micro-server Get Run Control thread depicted in FIG. 15. Only one HTTP connection thread is typically executed during a single invocation of the micro-server from the OEM software, with each subsequent execution processing the next active HTTP thread. The HTTP request could be retrieved from the TCP socket bound to port 80, as depicted at 1800. If the HTTP request is a valid GET request, the data cache could be checked for staleness, the data cache could be updated, if appropriate, and the HTML defining the requested document could be regenerated, as depicted at 1802, 1804, and 1806. Validation of a GET request could include validating the request format and the specified document. Using a standard file system paradigm, document pathnames could be resolved relative to internal micro-server data structures. Dynamic fields, such as the date, time, and an update counter could be updated, as shown at 1808.

If the document pathname specifies attributes of a micro-server variable, an alternate HTML document could be generated specifying the requested variable information, as depicted at 1810 and 1812. Otherwise, a standard response to the GET request could be generated including the requested document, as depicted at 1810 and 1814.

If the client request is a valid HEAD request, the HTML header information for the requested document could be constructed, as depicted at 1816 and 1818, and a complete HTTP response to the HEAD request could be generated, as depicted at 1820. The validation of HEAD requests can include validating the request format and the specified

document. Using the standard file system paradigm, document pathnames could be resolved relative to the internal micro-server data structures.

POST requests could be used to write data to the micro-server. If a valid POST request is received, the micro-server could internally simulate a CGI server to process the request, as depicted at 1822 and 1824. If the data being written by the client affects OEM data, as opposed to exposed micro-server interface data, the OEM data cache could be updated and the affected OEM data values could be communicated to the OEM software via callback functions specified during initialization, as indicated at 1826. The HTTP response could be generated to the POST request, as depicted at 1828. The HTTP response could be sent to the originating HTTP client, as depicted at 1830. As depicted at 1832, control is then returned to the micro-server main run thread shown in FIG. 15. In the event of an error, the micro-server could construct a standard HTML error response specifying the appropriate error number and description, as depicted at 1834.

Referring to FIG. 19, subscription processing could be entered from the micro-server System thread, as depicted at 1620. Subscription processing's function is to send updates to subscribing clients. Initially, the next subscription could be retrieved from the subscription table, as depicted at 1900. Subscribers could be identified by their IP address. If there are any remaining subscriptions to be processed, the subscription triggers could be retrieved, as depicted at 1902 and 1904. Triggers are typically specified by subscribing clients. For example, a trigger could be based on an alarm on a specified data variable or an elapsed time period. If none of the triggers for the current subscriber have been satisfied, processing could proceed to the next subscriber, as depicted at 1906 and 1900. The OEM data cache could be refreshed, if necessary, as depicted at 1908 and 1910. The response to the client request could be generated and sent to the client's IP address, as depicted at 1912 and 1914. As depicted at 1902 and 1916, once all subscriptions have been processed, control could be returned to the micro-server system thread, which is depicted in FIG. 16.

APPENDIX A—Micro-Server OEM API Definition

The information contained in this section is an alphabetically organized listing of the micro-server OEM API. This API forms an interface between the OEM software layer and the micro-server itself. The API consists of several functional groups. They are:

Micro-server initialization with constant information

Micro-server initialization with initial operating parameters

Micro-server initialization with system services data

Micro-server initialization with device read/write data

For each of the functions comprising the OEM API, there is a synopsis of the call including the appropriate declaration, a narrative description, description of arguments, the return value, and associated usage notes.

us_DeleteMaintLog()

Micro-Server Operation

Description

Deletes a message entry from the device maintenance log.

Synopsis

```
#include <userver.h>
```

```
int us_DeleteMaintLog(int message_id)
```

Arguments

The following arguments are passed to this function:
message_id message identifier obtained from a previous call to us_WriteMaintLog().

Return Value

This function returns an identifier upon success and -1 upon failure. In the event of failure, the external variable Us_error contains an appropriate error code. Error codes are defined in userver.h.

Usage Notes

This function is called to delete a text entry from the device non-volatile maintenance log. The entry would have been previously made by calling us_WriteMaintLog(). If non-volatile memory is not present in the Micro-server-enabled device and the Micro-server has been remotely configured with a name or IP address of a local maintenance server, the deletion is made from the server log via an appropriate TCP/IP transaction.

See Also

us_WriteMaintLog()

us_GetStatus()

Micro-Server Operation

Description

Returns current Micro-server status

Synopsis

```
#include <userver.h>
```

```
int us_GetStatus(void)
```

Arguments

none

Return Value

This function returns a 16-bit bitmask representing logically ORed Micro-server status. Normal operating status is 0. Individual bit flags are defined in userver.h.

Usage Notes

This call can be made in any context.

See Also

us_Run()

us_GetVariable()

Micro-Server Operation

Description

Returns a value of a variable

Synopsis

```
#include <userver.h>
```

```
char *us_GetVariable(char *var_name)
```

Arguments

The following arguments are passed to this function:
var_name Character string containing the name of the variable

Return Value

This function returns a pointer to the value of the requested variable. The pointer should be immediately used since it will become invalid with a next call to any API function. If the variable name passed to the function is invalid, this function returns a NULL pointer.

Usage Notes

This function is primarily used to retrieve values of variables set by the client, and its use is discouraged in

normal operation. Values of data variables cannot be retrieved via this mechanism, since their variable names are dynamically assigned by the Micro-server code during initialization and, in particular, since their values are already known to the OEM layer. Any attempt to retrieve data variable values by "synthesizing" their names will yield unpredictable results.

us_InitDone()

Micro-Server Initialization

Description

Concludes the Micro-server initialization process.

Synopsis

```
#include <userver.h>
int us_InitDone(void)
```

Arguments

None

Return Value

This function returns 0 (US_OK) if the initialization process has been performed correctly and -1 upon failure. In the event of failure, the external variable Us_error contains an appropriate error code. Error codes are defined in userver.h.

Usage Notes

Us_InitDone() should be the last of a set of calls performed to initialize the Micro-server. The server will not begin operation until the call is made with a value of US_OK returned to the caller.

us_InitMem()

Micro-Server Initialization

Description

Initializes the Micro-server memory

Synopsis

```
#include <userver.h>
int us_InitMem(char *memory, unsigned long memsize,
unsigned long disksize, char *directory,)
```

Arguments

The following arguments are passed to this function:

memory Points to the first byte of a memory block to be used for Micro-server data space. The caller should insure that the address is word-aligned.

memsize Specifies the size of the data area available to Micro-server, expressed in bytes.

disksize Specifies the amount of disk storage available for use by Micro-server. If the number is 0, Micro-server will not use disk I/O.

directory If the hardware environment in which Micro-server executes includes non-volatile disk storage, this argument should contain the path for the creation of Micro-server files, e.g. "c:\us_files". If this argument is a null string, and disksize is non-zero, Micro-server will create file(s) in the current directory.

Return Value

This function returns 0 (US_OK) upon success and -1 upon failure. In the event of failure, the external variable Us_error contains an appropriate error code. Error codes are defined in userver.h.

Usage Notes

If disk storage is not available in the execution environment, functions facilitated by non-volatile memory can still be used by making a call to us_Non Volatile().

See Also

us_NonVolatile()

us_Initstart()

Micro-Server Initialization

Description

Micro-server initialization process.

Synopsis

```
#include <userver.h>
int us_InitStart(void)
```

Arguments

None

Return Value

This function always returns 0 (US_OK)

Usage Notes

Us_InitStart() should be the first of a set of calls performed to initialize the Micro-server. If the call is made again, initialization starts all over.

us_Non Volatile()

Micro-Server Initialization

Description

Informs the Micro-server about non-volatile memory present in the execution environment.

Synopsis

```
#include <userver.h>
int us_NonVolatile(char memory, unsigned long memsize, void (*nvwrfunc)(), unsigned char (*nvrdfunc)())
```

Arguments

The following arguments are passed to this function:

memory Points to the first byte of a non-volatile memory block to be used for Micro-server non-volatile data.

memsize Specifies the size of the non-volatile data area available to Micro-server, expressed in bytes.

nvwrfunc Specifies the address to an OEM-supplied function, which returns a void, used to write a single byte of data to the nonvolatile data area. Micro-server assumes that the function is used as follows:

```
unsigned char written_data;
```

```
char *address;
```

```
nvwrfunc(address, written_data)
```

nvrdfunc Specifies the address to an OEM-supplied function, which returns an unsigned char, used to read a single byte of data from the nonvolatile data area. Micro-server assumes that the function is used as follows:

```
unsigned char read_data;
```

```
char *address;
```

```
read_data=nvrdfunc(address)
```

Return Value

This function returns 0 (US_OK) upon success and -1 upon failure. In the event of failure, the external variable Us_error contains an appropriate error code. Error codes are defined in userver.h.

Usage Notes

The OEM program should call NonVolatile() if disk storage is not available in the execution environment. If neither disk storage or non-volatile memory are available, Micro-server functions which require non-volatile memory will either not be available or will not be non-volatile.

The user program should define the `nvrdfunc()` and `nvwrfunc()`.

`us_Run()`

Micro-Server Operation

Description

Passes control the Micro-server

Synopsis

```
#include <userver.h>
int us_Run(void)
```

Arguments

none

Return Value

This function returns a 16-bit bitmask representing logically ORed Micro-server status. Normal operating status is 0. Individual bit flags are defined in `userver.h`.

Usage Notes

The purpose of `us_Run()` is to provide execution cycles to the Micro-server. This is the fundamental mechanism, which is used to operate the Micro-server. Under normal circumstances, the call to this function is made from within a loop in the OEM software layer. Alternately, the call can be made from an interrupt service routine for the real-time clock. The call is non-blocking, since the Micro-server adheres to "passthrough" architecture with no loops. Execution occurs on the stack in the current context.

See Also

`us_GetStatus()`

`us_SetHttpLatency()`

Micro-Server Initialization

Description

Sets up the Micro-server HTTP latency.

Synopsis

```
#include <userver.h>
int us_SetHttpLatency(long latency)
```

Arguments

The following arguments are passed to this function:
latency HTTP latency in milliseconds

Return Value

This function returns 0 (US_OK) upon success and -1 upon failure. In the event of failure, the external variable `Us_error` contains an appropriate error code. Error codes are defined in `userver.h`.

Usage Notes

This function is called to set the HTTP latency. The HTTP latency is defined as the maximum allowable age of the OEM data cache for HTTP get requests from the client. If the client browser issue a get request for either the read or the control page, and the data in the OEM data cache is older than the latency period, Micro-server will call the specified callback functions to obtain the current data from the OEM software layer, thus refreshing the cache. The OEM data cache timestamp will be updated accordingly. The default value for the HTTP latency is 5000 ms.

See Also

`us_SetTcpLatency()`

`us_SetTcpLatency()`

Micro-Server Initialization

Description

Sets up the Micro-server TCP/IP latency.

Synopsis

```
#include <userver.h>
int us_SetTcpLatency(long latency)
```

Arguments

The following arguments are passed to this function:
latency TCP/IP latency in milliseconds

Return Value

5 This function returns 0 (US_OK) upon success and -1 upon failure. In the event of failure, the external variable `Us_error` contains an appropriate error code. Error codes are defined in `userver.h`.

Usage Notes

10 This function is called to set the TCP/IP latency. The TCP/IP latency is defined as the maximum allowable age of the OEM data cache for TCP/IP requests from the client over the VTG application-to-application protocol. If the client software issues a request for any OEM data variables, and the data in the OEM data cache is older than the latency period, Micro-server will call the specified callback functions to obtain the current data from the OEM software layer, thus refreshing the cache. The OEM data cache timestamp will be updated accordingly. The default value for the TCP/IP latency is 100 ms.

See Also

`us_SetHttpLatency()`

`us_SendTcpResponse()`

Micro-Server Communication

Description

Used for TCP/IP dialog directly between remote client and user program, this function sends a response to a message originated by a remote TCP/IP client. This function is only used in the "passthrough" mode.

Synopsis

```
#include <userver.h>
int us_SendTcpResponse(IPADDRESS originator,
PACKET *packet)
```

Arguments

35 The following arguments are passed to this function:
originator Specifies the IP address of the originating client. This value will have been passed to the user code via the `tcpreceive()` function called by the Micro-server.

40 packet Pointer to the packet containing the response to send to the client

Return Value

This function returns 0 (US_OK) upon success and -1 upon failure. In the event of failure, the external variable `Us_error` contains an appropriate error code. Error codes are defined in `userver.h`.

See Also

`us_SetTcpDialog()`

`us_SetIpAddress()`

Micro-Server Initialization

Description

Micro-server IP address

Synopsis

```
55 #include <userver.h>
int us_SetIpAddress(IPADDRESS address)
```

Arguments

Address is a 32-bit entity containing the IP address of the embedded Micro-server. One byte is allocated to each of the four (4) address components with the most significant 8 bits of address corresponding to the most significant IP address component.

Return Value

65 This function returns 0 (US_OK) upon success and -1 upon failure. In the event of failure, the external variable `Us_error` contains an appropriate error code. Error codes are defined in `userver.h`.

Usage Notes

The value of the IP address passed by the user to Micro-server is typically read from the embedded hardware constituting the execution platform, such as DIP switches. Before initialization, the IP address is set to a default value of 100.100.100.1.

See Also

us_SetIpDHCP()

us_SetIpDHCP()

Micro-Server Initialization

Description

Initializes the Micro-server IP address via a remote DHCP server.

Synopsis

```
#include <userver.h>
int us_SetIpDHCP(void)
```

Arguments

None

Return Value

This function returns 0 (US_OK) upon success and -1 upon failure. In the event of failure, the external variable Us_error contains an appropriate error code. Error codes are defined in userver.h.

Usage Notes

The value of the IP address is determined by a remote DHCP server. This dynamic address assignment should only be used if the Micro-server-enabled device is accessed via a name resolved by a local DNS server.

See Also

us_SetIpAddress()

us_SetMfgData()

Micro-Server Initialization

Description

Used to provide the Micro-server with manufacturer data. Multiple calls to this function are made to provide such data. Majority of the data is destined for the /discover.htm page of the server.

Synopsis

```
#include <userver.h>
int us_SetMfgData(int id, char *value)
```

Arguments

The following arguments are passed to this function:

id Identifies the data passed in the value argument to the function. Header file "userver.h" contains manifest constants used for this value. Because these constants contain additional implicit information, they should always be used. Do not use constant values here. For example, US_MFGADR0 should be used instead of 2.

value Specifies the value of the field identified by id. The format of this argument is always a null-terminated string.

Return Value

This function returns 0 (US_OK) upon success and -1 upon failure. In the event of failure, the external variable Us_error contains an appropriate error code. Error codes are defined in userver.h.

Usage Notes

The following table specifies constant values that could be used with the id argument

US_CAPLTSVR	Local Applet Server name
US_CMNTSVR	Maintenance Server name

-continued

US_CMYIPADRS	Device IP address
US_CTIMSVR	Local time server IP name
US_ICOUNTRY	Country code
US_MFGADRS0	Manufacturer address 0
US_MFGADRS1	Manufacturer address 1
US_MFGADRS2	Manufacturer address 2
US_MFGADRS3	Manufacturer address 3
US_MFGAPLURL	Manufacturer client applet server URL
US_MFGDATE	Device date of manufacture
US_MFGDOCURL	Manufacturer documentation URL
US_MFGEMAIL	Manufacturer tech support Email address
US_MFGFAX	Manufacturer fax number
US_MFGGENER	Manufacturer generic device description
US_MFGLNKMAIL	Manufacturer tech support Email link
US_MFGLOGO	Link to Manufacturer logo
US_MFGMODEL	Manufacturer device model number
US_MFGMSID	Manufacturer Micro-server License ID
US_MFGNAME	Manufacturer Name
US_MFGPGMVER	Manufacturer Program Version
US_MFGSERNO	Manufacturer device serial number
US_MFGTEL	Manufacturer telephone number
US_MFGURL	Manufacturer general URL
US_MGGBACK	Manufacturer background

us_SetReadData()

Micro-Server Initialization

Description

Micro-server with data to be published to the network. The data is referred to as "read" since it is read by the client.

Multiple calls to this function are made to provide such data, with one call made for each of the published data points. Majority of the data is destined for the /read.htm or /default.htm page of the server.

Synopsis

```
#include <userver.h>
int us_SetReadData(READDATA *ptr)
```

Arguments

The following arguments are passed to this function:

ptr Is a pointer to a READDATA definition structure initialized by the caller before the call to us_SetReadData(). Header file "userver.h" contains a declaration for this structure.

Return Value

This function returns 0 (US_OK) upon success and -1 upon failure. In the event of failure, the external variable Us_error contains an appropriate error code. Error codes are defined in userver.h.

Usage Notes

The calling program passes a pointer to a data structure containing the description of the read data field to be published. The data structure is declared as:

```
typedef struct readdata
{
    char *description; /* describes the parameter */
    char *units; /* specifies the measurement units */
    char (*callback)(); /* call back function used to get value */
    char *lowend; /* the low end of parameter value span */
    char *highend; /* the low end of parameter value span */
    char *lowlimit; /* recommended low limit for parameter value */
    char *highlimit; /* recommended high limit for parameter value */
    char *nominal; /* recommended nominal value for parameter */
} READDATA;
```

The callback element of the READDATA structure contains a pointer to a user-provided function that returns the current value of the data. The data is returned as a character string.

Note that the lowlimit, highlimit and nominal strings are advisory in nature. If the user program chooses not to specify them, they may be left blank (by assigning 0-length null terminated strings to corresponding structure elements. The Micro-server makes private copies of the user data, hence the READDATA structure may be re-used with a subsequent call to us_SetReadData().

See Also

us_SetReadData()

us_SetWriteData()

Micro-Server Initialization

Description

Configures the Micro-server with control data to be published and accepted from the clients. The data is referred to as "write" since it can be written to the OEM layer by the remote client. Multiple calls to this function are made to provide such data, with one call made for each of the published data points. Majority of the data is destined for the /control.htm page of the server.

Synopsis

```
#include <userver.h>
int us_SetWriteData(WRITEDATA *ptr)
```

Arguments

The following arguments are passed to this function:

ptr Is a pointer to a WRITEDATA definition structure initialized by the caller before the call to us_SetWriteData(). Header file "userver.h" contains a declaration for this structure.

Return Value

This function returns 0 (US_OK) upon success and -1 upon failure. In the event of failure, the external variable Us_error contains an appropriate error code. Error codes are defined in userver.h.

Usage Notes

The calling program passes a pointer to a data structure containing the description of the read data field to be published. The data structure is declared as:

```
typedef struct readdata
{
    char *description; /* describes the parameter */
    char *units; /* specifies the measurement units */
    char (*rdcallback)(); /* call back function used to get value */
    char (*wrcallback)(); /* call back function used to set value */
    char *lowend; /* the low end of parameter value span */
    char *highend; /* the low end of parameter value span */
    char *lowlimit; /* recommended low limit for parameter value */
    char *highlimit; /* recommended high limit for parameter value */
    char *nominal; /* recommended nominal value for parameter */
} READDATA;
```

The callback element of the WRITEDATA structure contains a pointer to a user-provided function that returns the current value of the data. The data is always passed as a character string.

Note that the lowlimit, highlimit and nominal strings are advisory in nature. If the user program chooses not to specify them, they may be left blank (by assigning 0-length null terminated strings to corresponding structure elements.

The Micro-server makes private copies of the user data, hence the WRITEDATA structure may be re-used with a subsequent call to us_SetWriteData().

See Also

us_SetReadData()

us_SetTcpDialog()

Micro-Server Initialization

Description

Initializes TCP/IP dialog directly between remote client and user program. This is necessary only in the "passthrough" mode.

Synopsis

```
#include <userver.h>
int us_SetTcpDialog(int port, void (*tcpreceive)())
```

Arguments

The following arguments are passed to this function:

portid Specifies the port number to be used in direct communications with the client. Keep in mind that the port may not conflict with the two primary ports are already used: US_SERVERPORT used by the Micro-server for handling HTTP protocol traffic as well US_TCPPORT used by the Micro-server for handling transactions involving client-side generic applets.

tcpreceive Specifies the address to an OEM function of type void, used to write pass a TCP packet received by Micro-server to the OEM software layer. Whenever a TCP packet is received on the specified port, Micro-server calls this function to pass the packet to the OEM software.

Micro-server assumes that the function is used as follows:

```
IPADDRESS originator;
PACKET *packet;
tcpreceive(originator, packet)
```

Return Value

This function returns 0 (US_OK) upon success and -1 upon failure. In the event of failure, the external variable Us_error contains an appropriate error code. Error codes are defined in userver.h.

Usage Notes

The user program is responsible for providing the tcpreceive() function if this feature will be used.

See Also

us_SendTcpResponse()

us_WriteMaintLog()

Micro-Server Operation

Description

Micro-server has been remotely configured with a name or IP address of a local maintenance server, the entry is made in the server log via an appropriate TCP/IP transaction.

See also

us_deleteMaintLog()

We claim:

1. A system for providing information about a remote device to a client workstation, the system comprising:

a micro-server for transmitting the information to the client workstation, the micro-server defining an application programming interface for interfacing with the remote device to access the information from the remote device and for abstracting the details of transmitting the information to the client workstation, the remote device initializing the micro-server by providing, via a function call defined by the application programming interface, at least one pointer to at least one callback function for accessing the information, the

31

micro-server accessing the information by calling the at least one callback function.

2. The system of claim 1 wherein the micro-server comprises a TCP/IP application programming interface for accessing a TCP/IP protocol stack.

3. The system of claim 1 further comprising: a system services application programming interface for providing the micro-server with access to system services from the remote device.

4. The system of claim 1 wherein the micro-server comprises an HTTP protocol server for satisfying interactive HTTP requests from the remote client workstation.

5. The system of claim 4 wherein the micro-server comprises a synthesized HTML server for providing a dynamic copy of an HTML version of a web page served by the HTTP protocol server.

6. The system of claim 1 further comprising: a data image for caching data from the remote device for reducing the number of application programming interface function calls made by the micro-server to access remote device information.

7. The system of claim 1 wherein the micro-server comprises: a TCP/IP based server for satisfying TCP/IP based requests from the client workstation.

8. The system of claim 1 wherein the micro-server comprises: a hyperlink to a website associated with the remote device.

9. The system of claim 1 wherein the micro-server comprises: a web site associated with the remote device.

10. The system of claim 9 wherein the web site comprises: a home page for publishing the remote device's parametric data.

11. The system of claim 9 wherein the web site comprises: a control page for providing authorized individuals access to the remote device's control functions.

12. The system of claim 9 wherein the web site comprises: a maintenance page for providing access to the remote device's maintenance data.

13. The system of claim 12 wherein the maintenance page comprises: a hyperlink to online maintenance documentation.

14. The system of claim 9 wherein the web site comprises: a maintenance page for providing access to the remote device's maintenance functions.

15. The system of claim 9 wherein the web site comprises: an administration page for allowing authorized individuals to set operating characteristics of the remote device.

16. The system of claim 9 wherein the web site comprises: a discover page for supplying information to the client workstation during automatic discovery.

17. The system of claim 1 further comprising: a default applet server for providing default applets to the client workstation for interfacing with the remote device from the client workstation.

18. The system of claim 1 further comprising: a time server for providing current time information to the system, the time information being maintained based at least in part upon universal coordinated time.

19. The system of claim 1 further comprising: a maintenance server for providing non-volatile storage for the remote device's maintenance records, the non-volatile storage being hyperlinked from the remote device.

20. The system of claim 1 further comprising: an auto-discovery and view server for automatically detecting a micro-server-enabled device being coupled to an interface and publishing micro-server-enabled device data to the client workstation.

32

21. The system of claim 1 further comprising: a local applet server for retrieving device-specific applets from a server associated with the remote device and for making the applets available to the client workstation.

22. The system of claim 1 further comprising: a browser for interacting with the remote device.

23. The system of claim 1 further comprising: a client-side application programming interface for providing the client workstation with a software interface to the remote device.

24. The system of claim 1 wherein at least a portion of the information is organized as addressable variables.

25. A remote device capable of providing information about itself to a client workstation, the remote device comprising:

a micro-server for transmitting the information to the client workstation while abstracting communication protocol details from the remote device's control software;

an application programming interface for providing an interface between the remote device's control software and the micro-server, the control software initializing the micro-server by providing, via a function call defined by the application programming interface, at least one pointer to at least one callback function for accessing the information, the micro-server accessing the information by calling the at least one callback function;

a hardware Ethernet interface; and

a TCP/IP protocol stack for interfacing between the micro-server and the hardware Ethernet interface.

26. The remote device of claim 25 further comprising: a computer capable of supporting a standard operating system environment.

27. A micro-circuit board for providing information about a remote device to a client workstation, the micro-circuit board comprising:

a micro-server for transmitting the information to the client workstation while abstracting communication protocol programming details from the remote device's control software;

an application programming interface for providing an interface between the remote device's control software and the micro-server, the control software initializing the micro-server by providing, via a function call defined by the application programming interface, at least one pointer to at least one callback function for accessing the information the micro-server accessing the information by calling the at least one callback function;

a hardware Ethernet interface; and

a TCP/IP protocol stack for interfacing between the micro-server and the hardware Ethernet interface.

28. A system for providing information about a remote device to a client workstation, the system comprising:

a first processor for running the remote device's control software and a first side of an application programming interface ("API");

a second processor for running micro-server software and abstracting communication protocol details from the first processor by running a second side of the API, the control software initializing the micro-server by providing, via a function call defined by the API, at least one pointer to at least one callback function for accessing the information, the micro-server accessing

33

the information by calling the at least one callback function and transmitting the information to the client workstation, the second processor accessing a TCP/IP stack to interface with the hardware Ethernet interface; and

a hardware interface between the first processor and the second processor.

29. A system for providing information about a remote device to a client workstation, the system comprising:

a processor for running remote device control software and a first side of an application programming interface ("API"), the processor being mounted on a PC circuit board, the PC circuit board being insertable into a computer;

a computer for abstracting communication protocol details from the processor by running micro-server software including a second side of the API, the control software initializing the micro-server by providing, via a function call defined by an application programming interface, at least one pointer to at least one callback function for accessing the information, the micro-server accessing the information by calling the at least one callback function and transmitting the information to the client workstation, the computer having a hardware Ethernet interface and a TCP/IP stack for interfacing with the hardware Ethernet interface; and

a hardware interface between the processor and the computer.

30. A method for providing information about a remote device to a client workstation comprising the steps of:

providing to a micro-server, via a function call defined by an application programming interface, at least one pointer to at least one callback function;

accessing the information from the first device via the at least one callback function;

organizing the information into a format compatible with a communication protocol in preparation for making the information available to the client workstation;

making the information available to the client workstation; and

34

abstracting the communication protocol from the remote device.

31. A system for accessing remote device data and communicating the data to a client workstation, the system comprising:

a remote device having an original equipment manufacturer ("OEM") software component for controlling operation of the remote device and a micro-server software component;

the micro-server software component for transmitting the remote device data to the client workstation, the micro-server software component having an original equipment manufacturer application programming interface ("OEM API") for allowing the OEM software to initialize the micro-server software, the initialization including the OEM software providing one or more callback functions to the micro-server software component to allow the micro-server software component to access OEM software component data through one or more functions defined by the OEM API, the OEM API abstracting micro-server software component implementation of networking protocol details from the OEM software component.

32. The system of claim 31 wherein the micro-server software component comprises non-blocking threads for returning processor control to the OEM software component without delays associated waiting for external events to occur.

33. The system of claim 31 wherein the micro-server software component is a binary software library linked to the OEM application.

34. The system of claim 31 wherein the micro-server software component includes a client-side refresh mode of operation that automatically updates a page displayed at the client workstation at fixed time intervals.

35. The system of claim 31 wherein the micro-server software component comprises an OEM data cache component for caching OEM data accessed by the micro-server software component thereby reducing overhead associated with frequent OEM API function calls.

* * * * *



US006581088B1

(12) **United States Patent**
Jacobs et al.

(10) **Patent No.:** **US 6,581,088 B1**
(45) **Date of Patent:** **Jun. 17, 2003**

(54) **SMART STUB OR ENTERPRISE JAVA™
BEAN IN A DISTRIBUTED PROCESSING
SYSTEM**

(75) Inventors: **Dean B. Jacobs, Berkeley, CA (US);
Eric M. Halpern, San Francisco, CA
(US)**

(73) Assignee: **Beas Systems, Inc., San Jose, CA (US)**

(*) Notice: Subject to any disclaimer, the term of this
patent is extended or adjusted under 35
U.S.C. 154(b) by 0 days.

(21) Appl. No.: **09/405,260**

(22) Filed: **Sep. 23, 1999**

Related U.S. Application Data

(60) Provisional application No. 60/107,167, filed on Nov. 5,
1998.

(51) Int. Cl.⁷ **G06F 9/00**

(52) U.S. Cl. **709/105; 709/100; 709/203**

(58) Field of Search **709/100, 102,
709/103, 104, 107, 108, 310, 315, 316,
317, 328, 318, 319**

(56) References Cited

U.S. PATENT DOCUMENTS

5,465,365 A 11/1995 Winterbottom 707/104
5,475,819 A 12/1995 Miller et al. 395/200.03
5,564,070 A 10/1996 Want et al. 455/53.1
5,623,666 A 4/1997 Pike et al. 707/200
5,692,180 A 11/1997 Lee 707/10
5,701,484 A 12/1997 Artsy 709/242
5,701,502 A 12/1997 Baker et al. 709/201
5,764,982 A 6/1998 Madduri 707/10
5,774,660 A * 6/1998 Brendel et al. 395/200.31
5,790,548 A 8/1998 Sistanizadeh et al. 707/10
5,794,006 A 8/1998 Sanderman 707/10
5,805,804 A 9/1998 Laursen et al. 395/200.02
5,819,019 A 10/1998 Nelson 707/10
5,819,044 A 10/1998 Kawabe et al. 395/200.56
5,842,219 A 11/1998 High, Jr. et al. 707/103
5,850,449 A 12/1998 McManis 703/161
5,862,331 A 1/1999 Herriot 395/200.49

5,901,227 A 5/1999 Perlman 380/21
5,961,582 A 10/1999 Gaines 709/1
5,966,702 A 10/1999 Fresko et al. 707/1
5,974,441 A 10/1999 Rogers et al. 709/200
5,983,233 A 11/1999 Potonniee 707/103
5,983,351 A 11/1999 Glogau 713/201
5,999,988 A 12/1999 Pelegri-Llopert et al. ... 709/330
6,003,065 A 12/1999 Yan et al. 709/201
6,006,264 A * 12/1999 Colby et al. 709/226
6,016,505 A 1/2000 Badovinatz et al. 709/205
6,144,944 A * 11/2000 Kurtzman, II et al. 705/14

OTHER PUBLICATIONS

Brewing Distributed Applications With Java ORB Tools,
Dec. 1996.*

Sun Microsystems, JavaBeans, Graham Hamilton, Jul.
1997.*

* cited by examiner

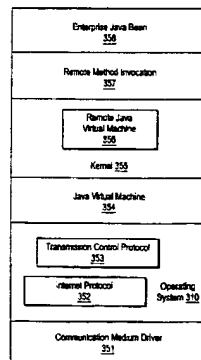
Primary Examiner—Majid Banankhah

(74) Attorney, Agent, or Firm—Fliesler, Dubb, Meyer &
Lovejoy LLP

(57) ABSTRACT

A clustered enterprise Java™ distributed processing system is provided. The distributed processing system includes a first and a second computer coupled to a communication medium. The first computer includes a Java™ virtual machine (JVM) and kernel software layer for transferring messages, including a remote Java™ virtual machine (RJVM). The second computer includes a JVM and a kernel software layer having a RJVM. Messages are passed from a RJVM to the JVM in one computer to the JVM and RJVM in the second computer. Messages may be forwarded through an intermediate server or rerouted after a network reconfiguration. Each computer includes a Smart stub having a replica handler, including a load balancing software component and a failover software component. Each computer includes a duplicated service naming tree for storing a pool of Smart stubs at a node. The computers may be programmed in a stateless, stateless factory, or a stateful programming model. The clustered enterprise Java™ distributed processing system allows for enhanced scalability and fault tolerance.

74 Claims, 16 Drawing Sheets



CLIENT/SERVER
ARCHITECTURE 110

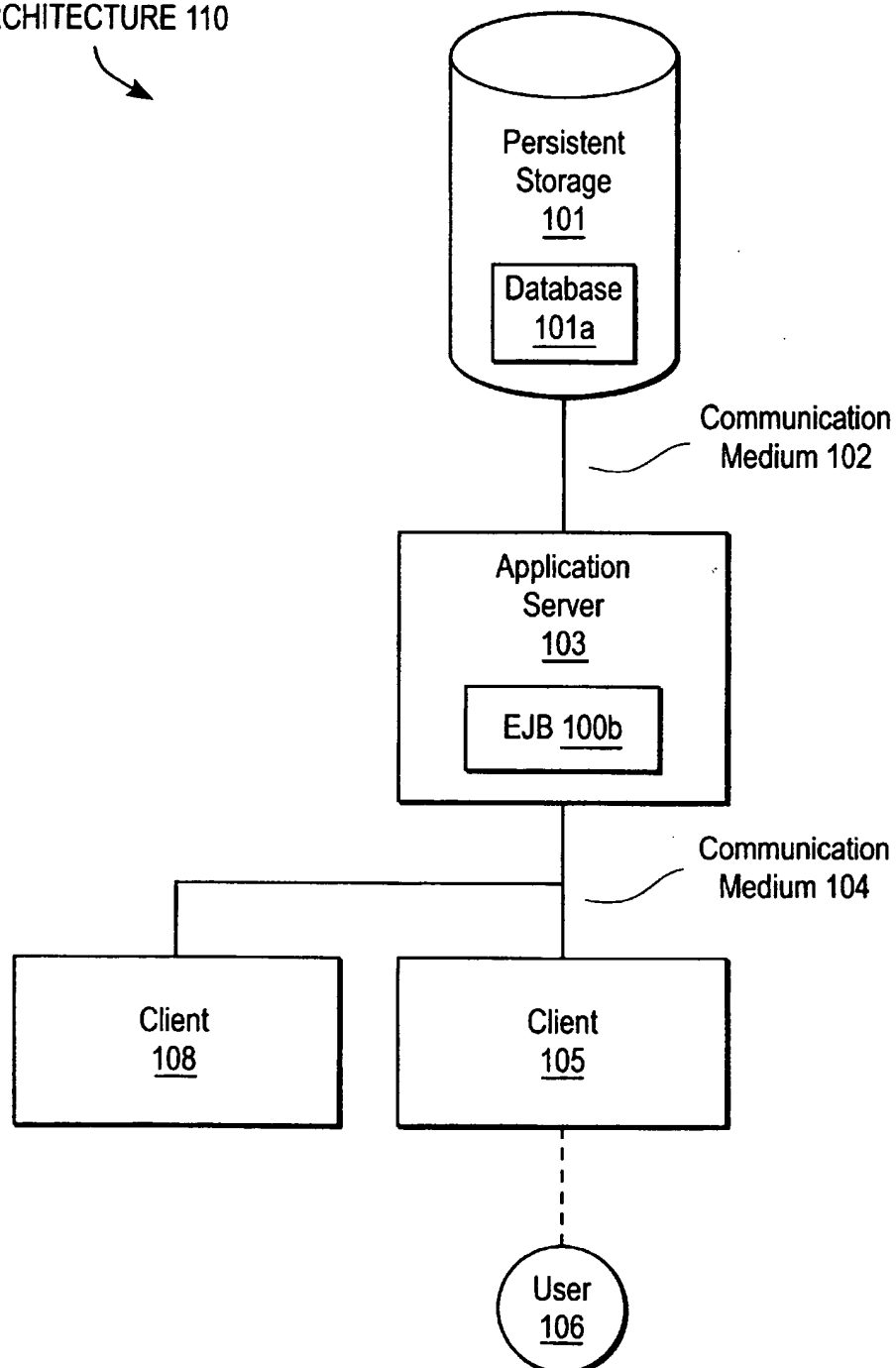


FIG. 1a (PRIOR ART)

JAVA ENTERPRISE APIs 100

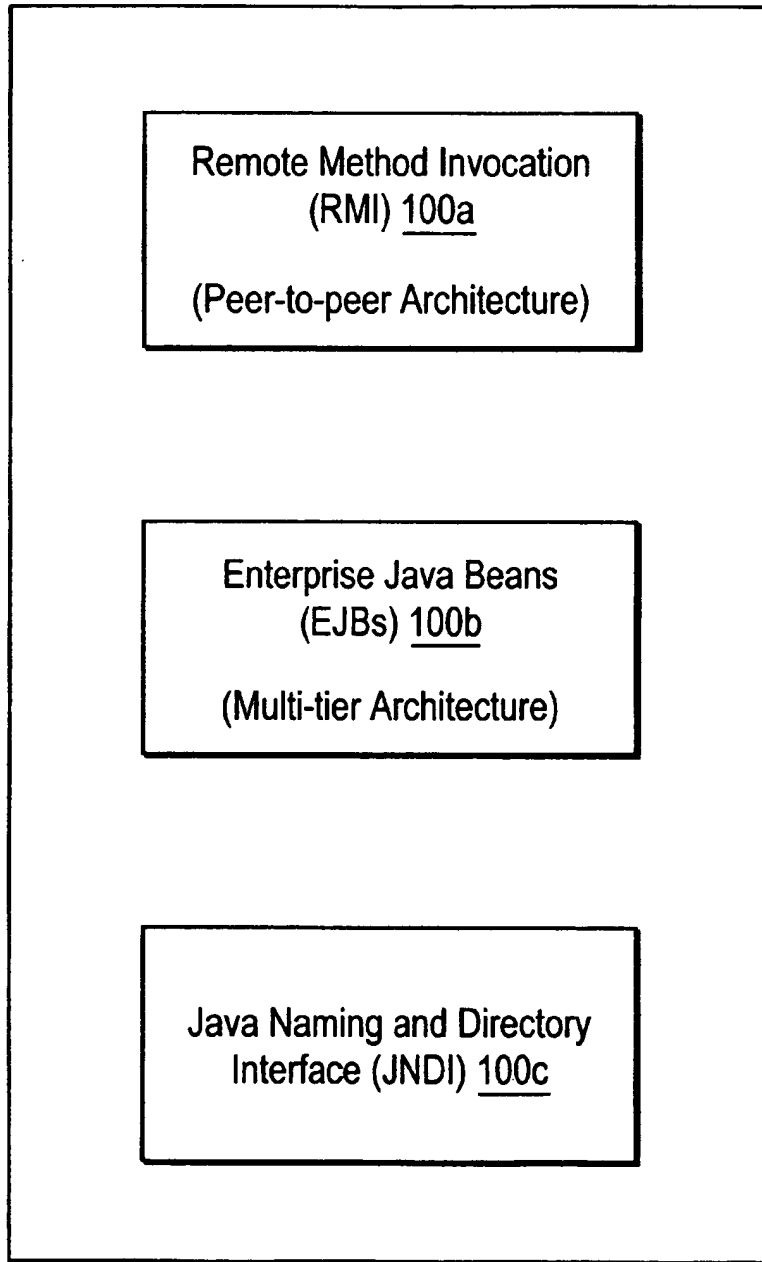


FIG. 1b (PRIOR ART)

MULTI-TIER
ARCHITECTURE 160

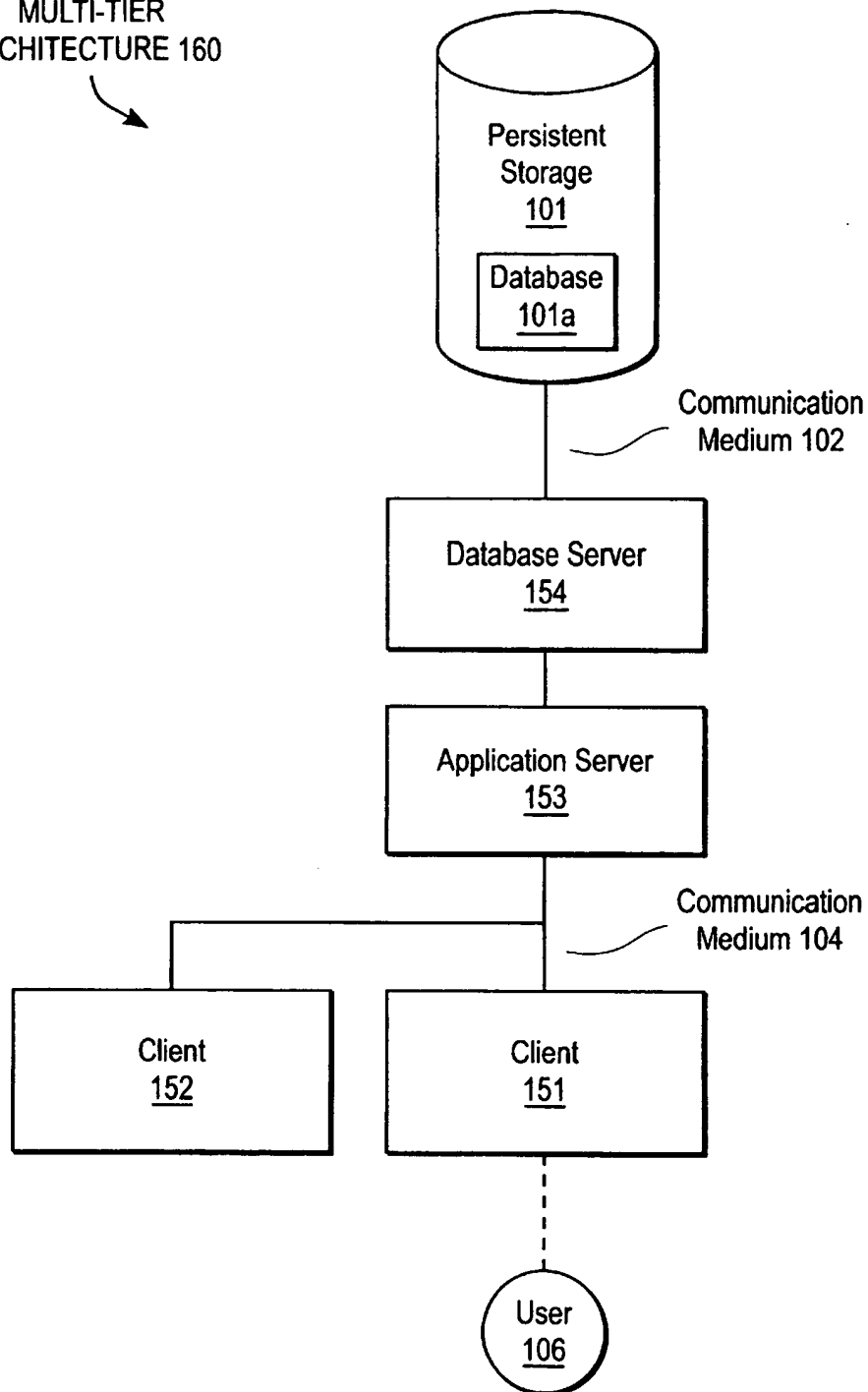


FIG. 1c (PRIOR ART)

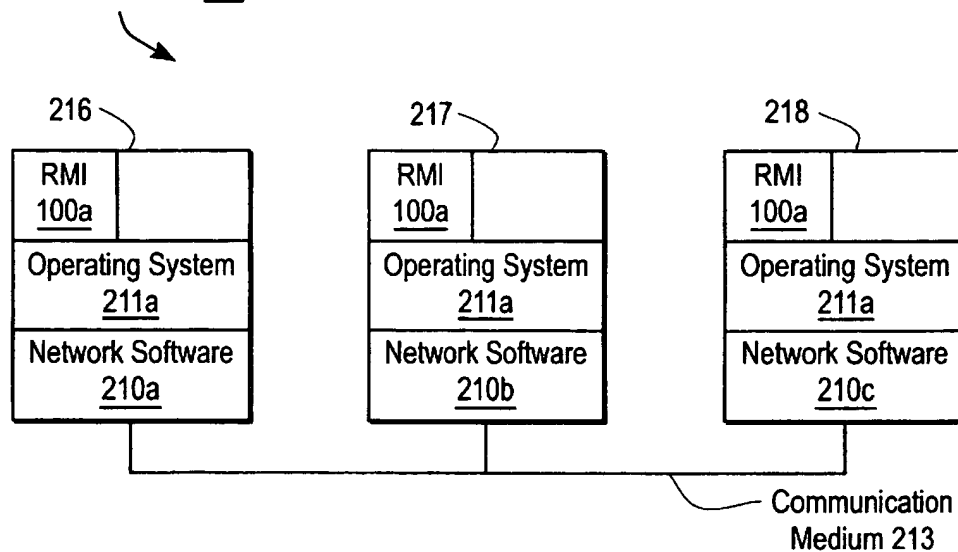
PEER-TO-PEER
ARCHITECTURE 214

FIG. 2a (PRIOR ART)

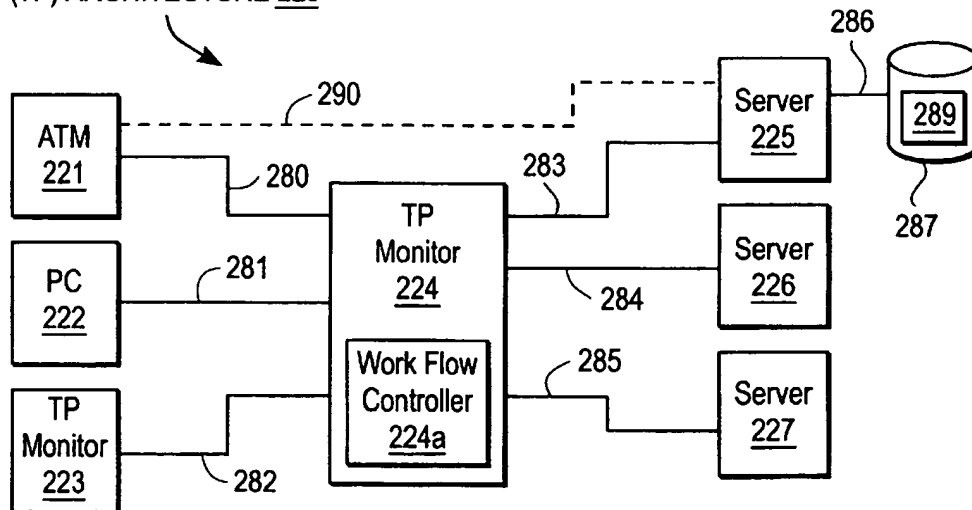
TRANSACTION PROCESSING
(TP) ARCHITECTURE 220

FIG. 2b (PRIOR ART)

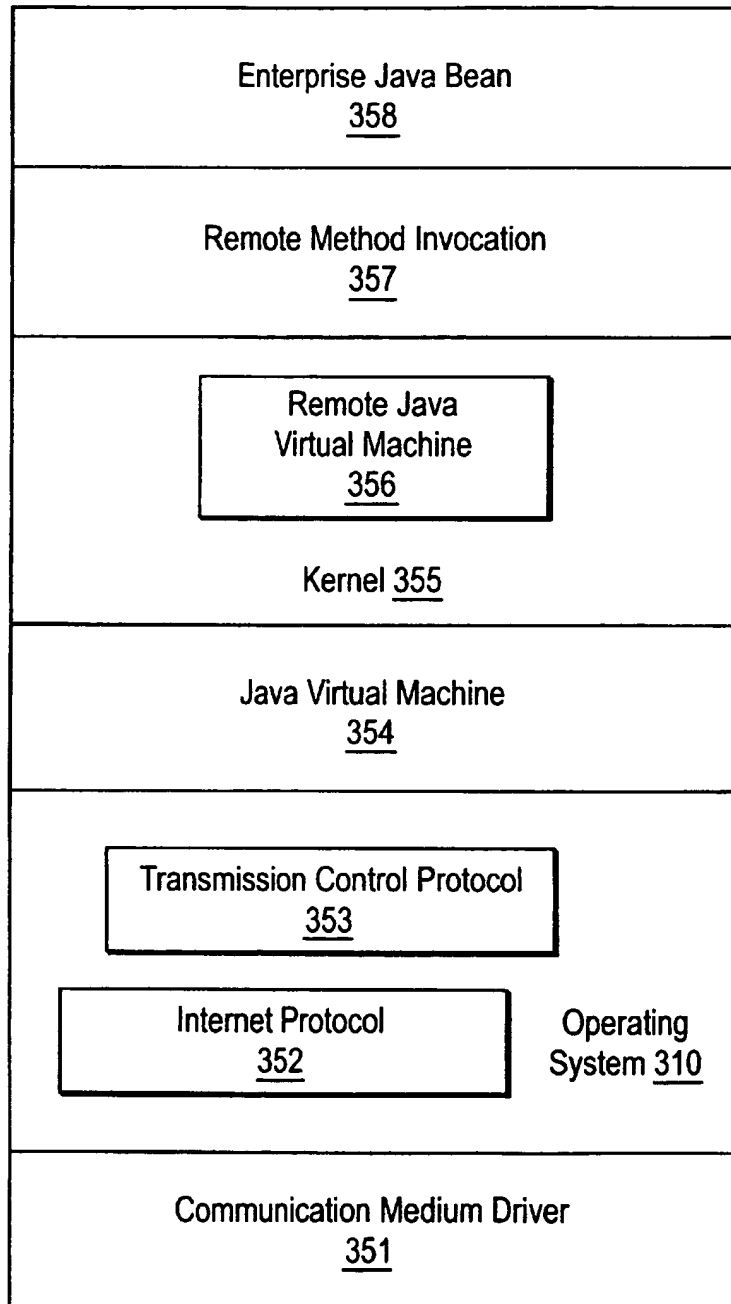
380
↘

FIG. 3a

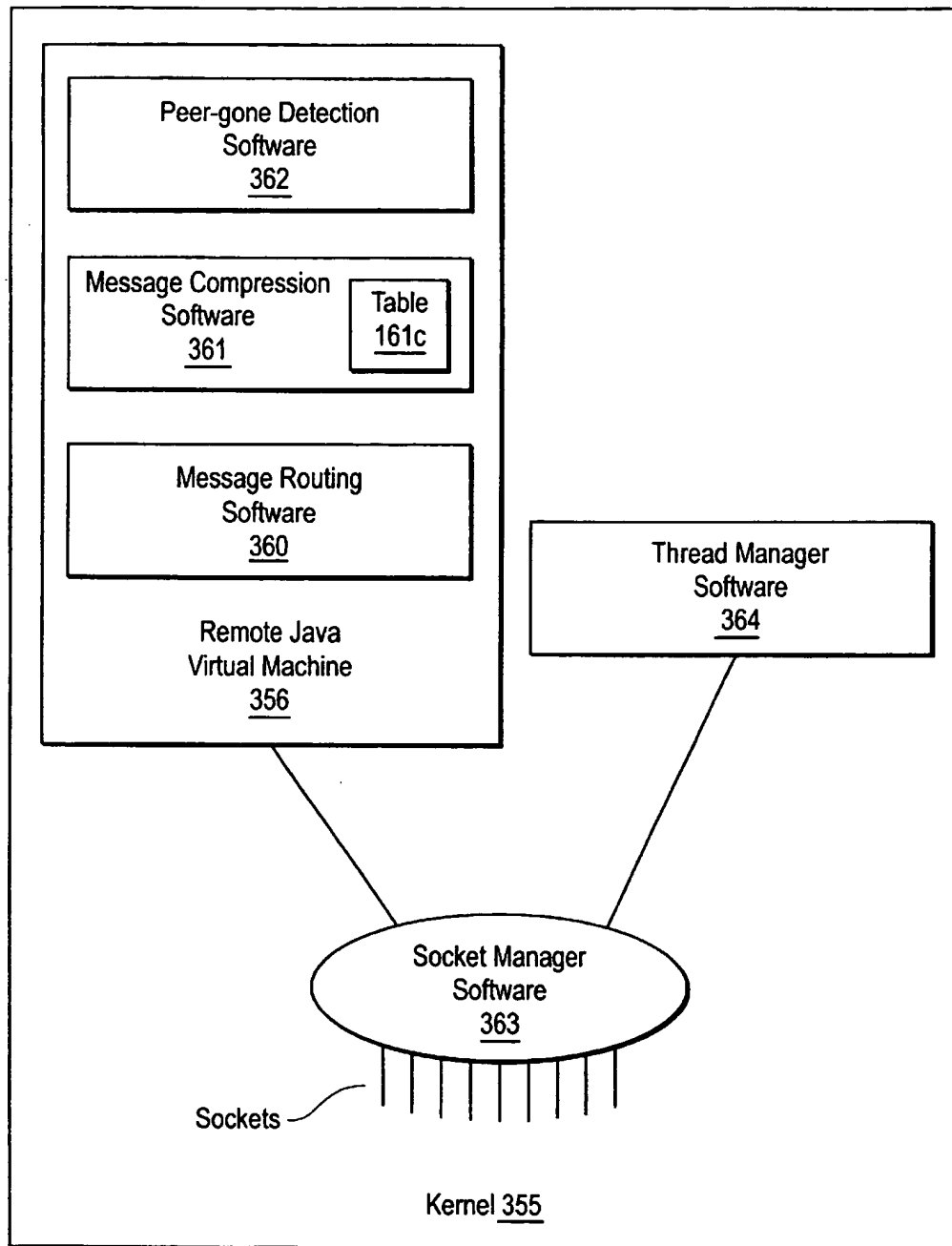


FIG. 3b

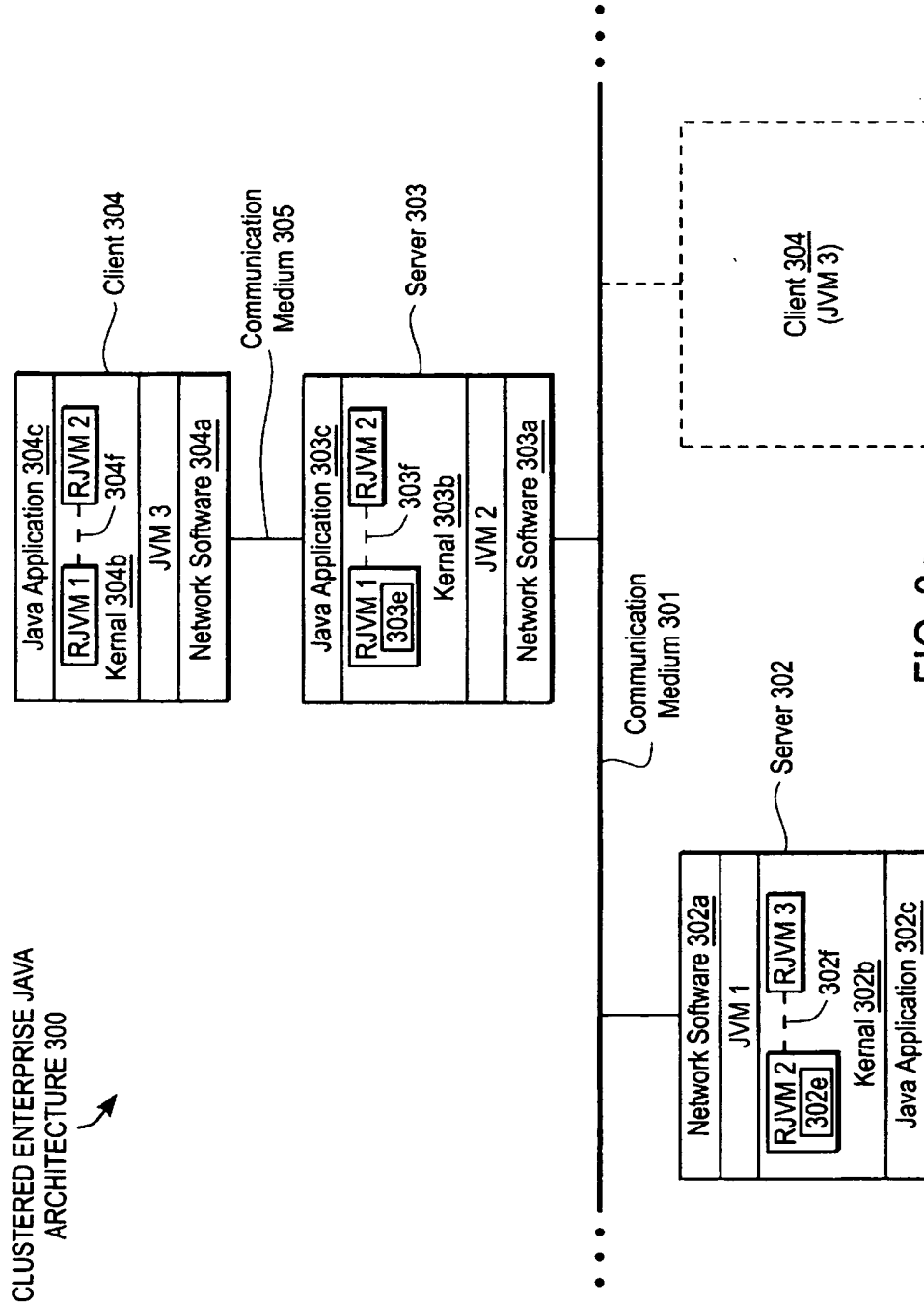


FIG. 3c

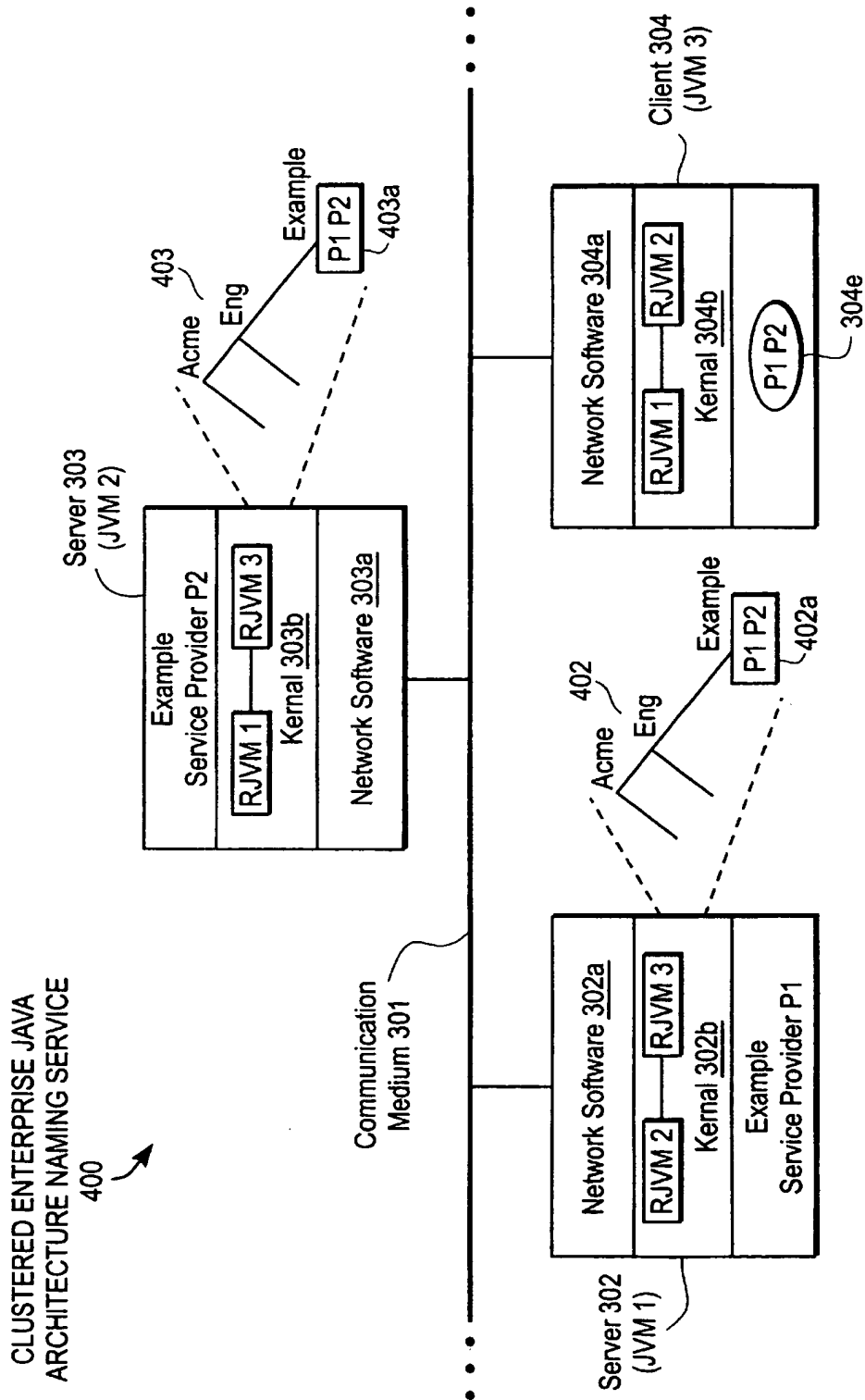


FIG. 4

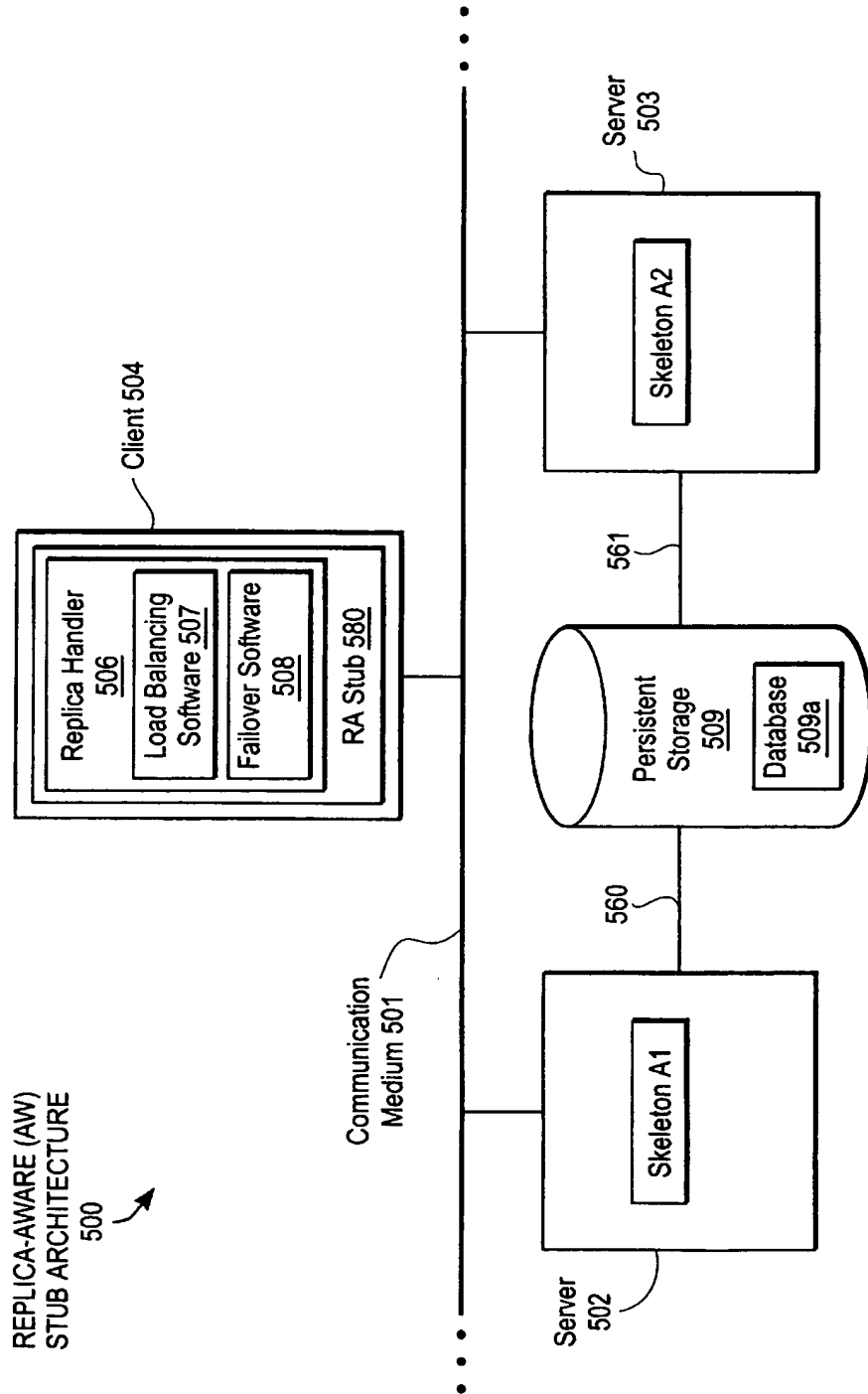


FIG. 5a

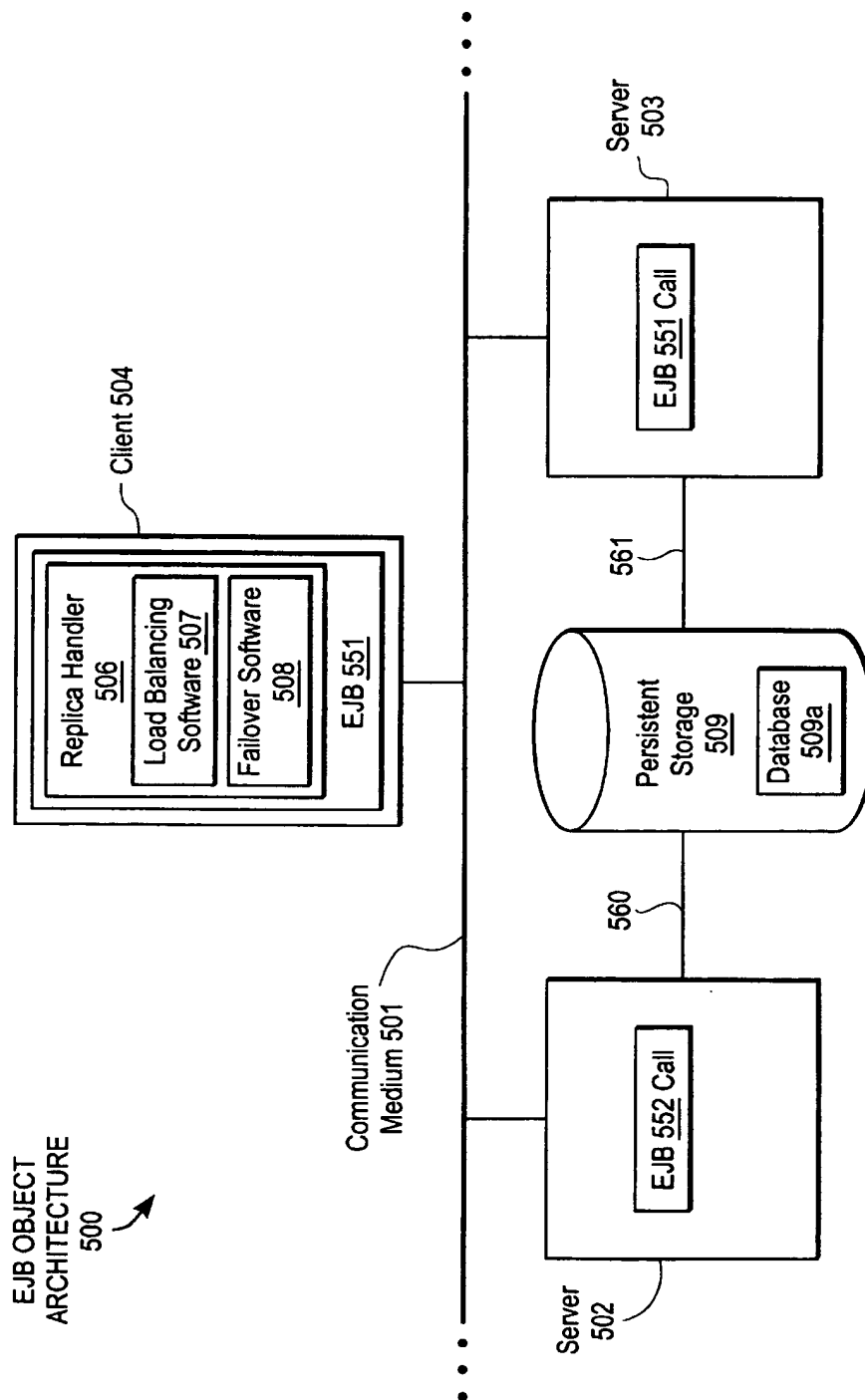


FIG. 5b

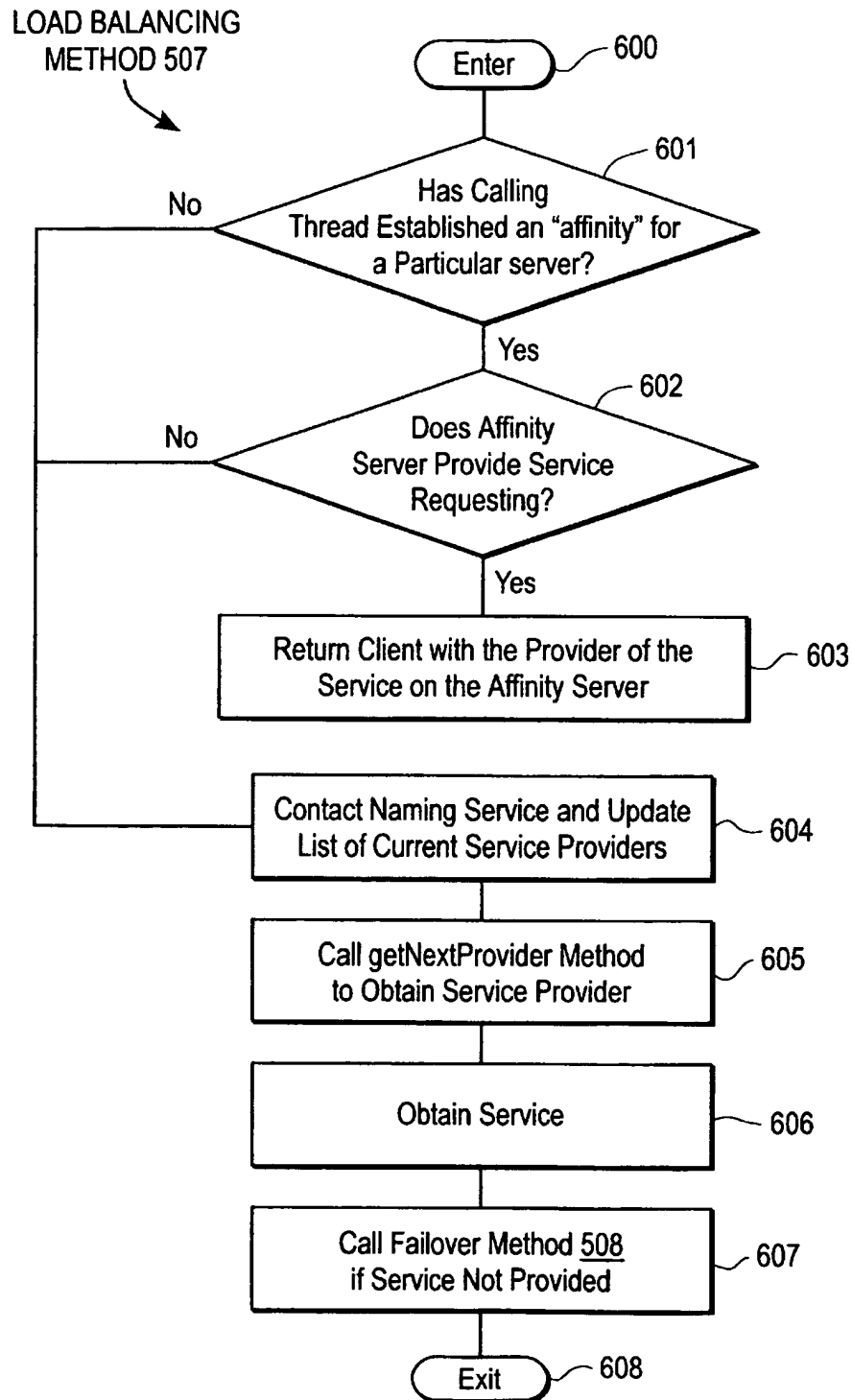


FIG. 6a

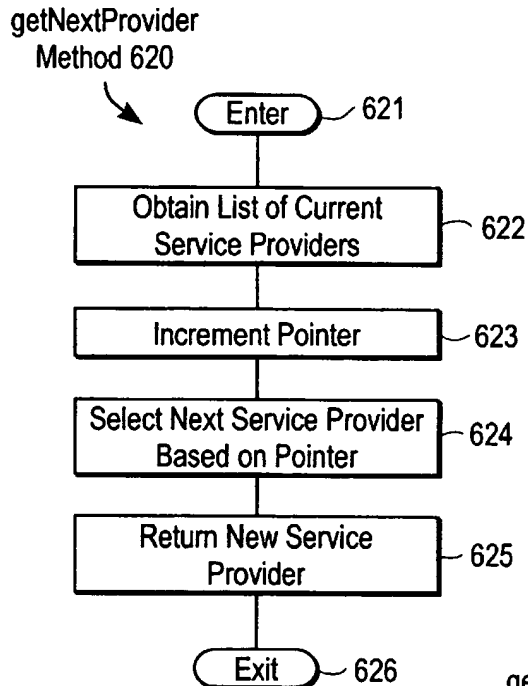


FIG. 6b

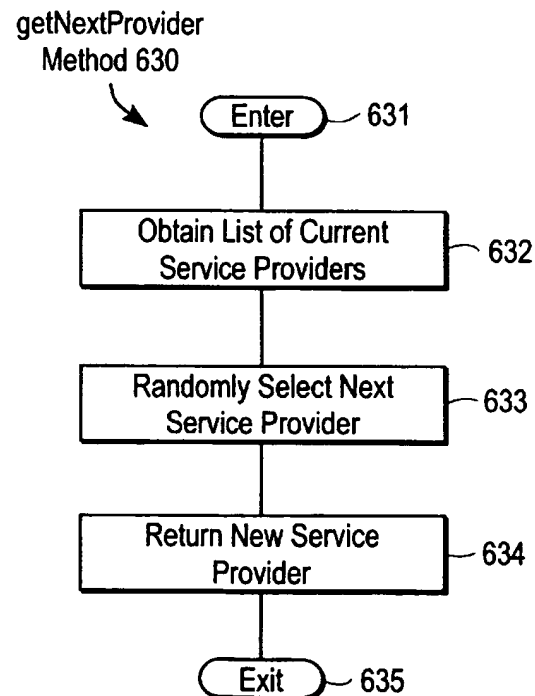


FIG. 6c

getNextProvider
Method 640

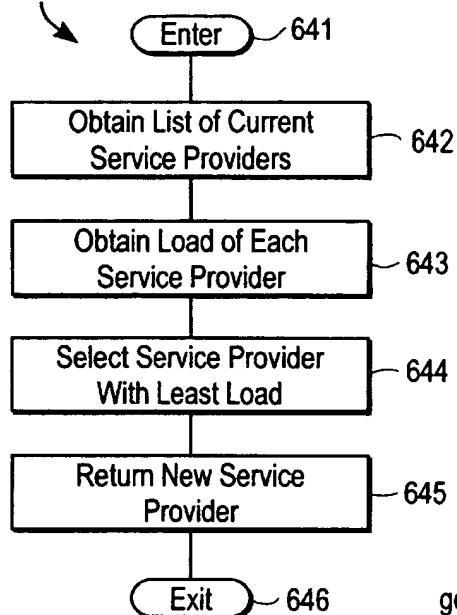


FIG. 6d

getNextProvider
Method 650

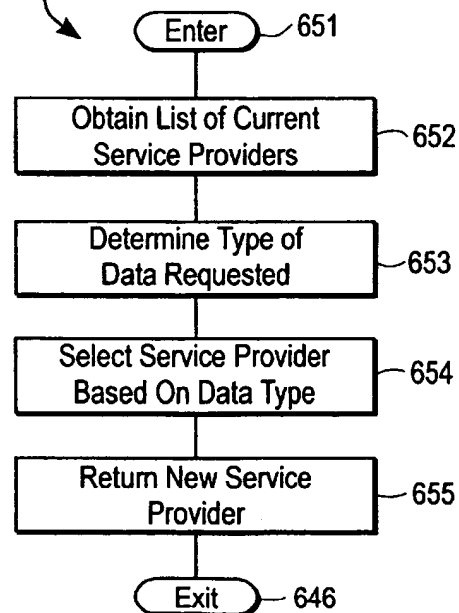


FIG. 6e

getNextProvider
Method 660

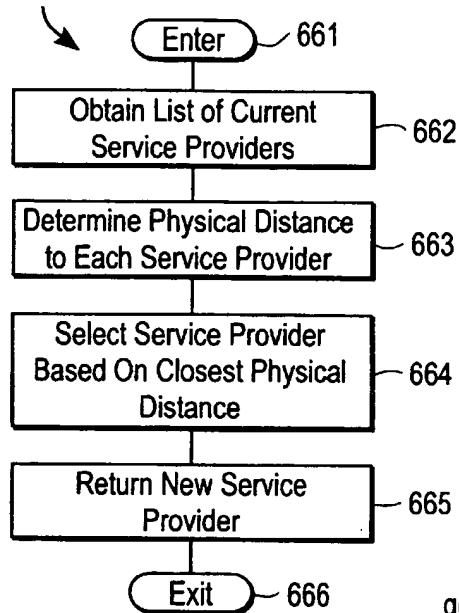


FIG. 6f

getNextProvider
Method 670

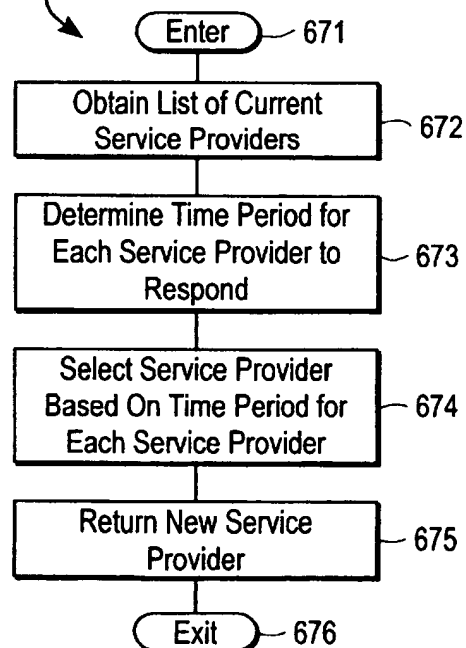


FIG. 6g

FAILOVER METHOD

508

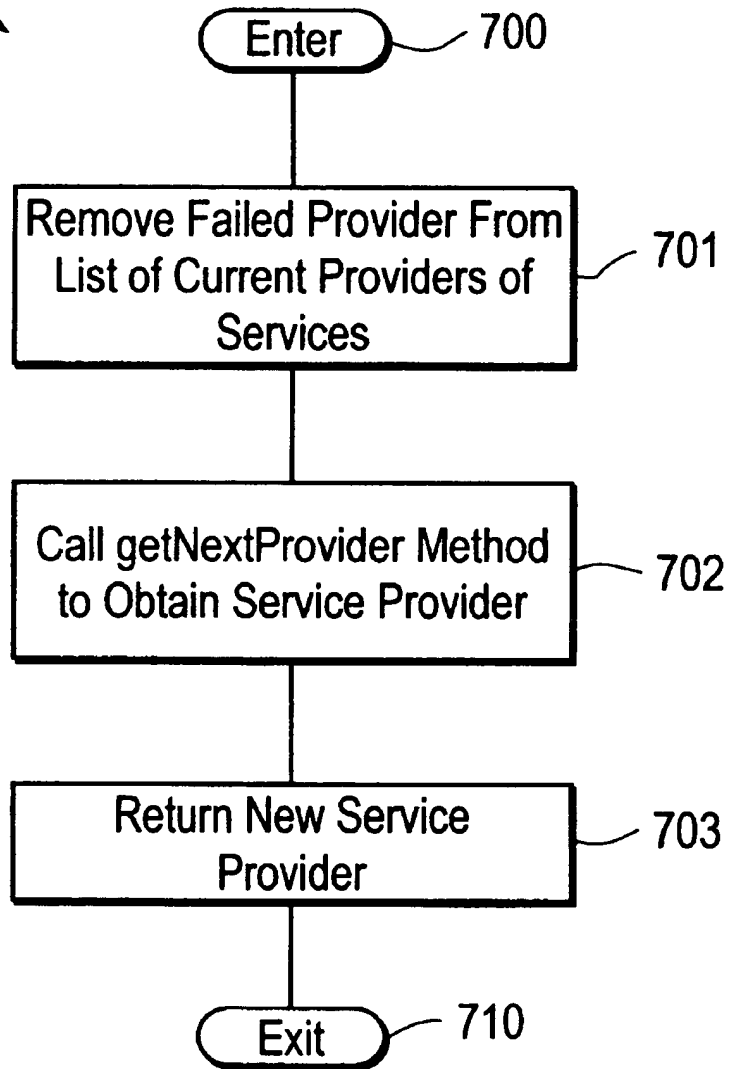


FIG. 7

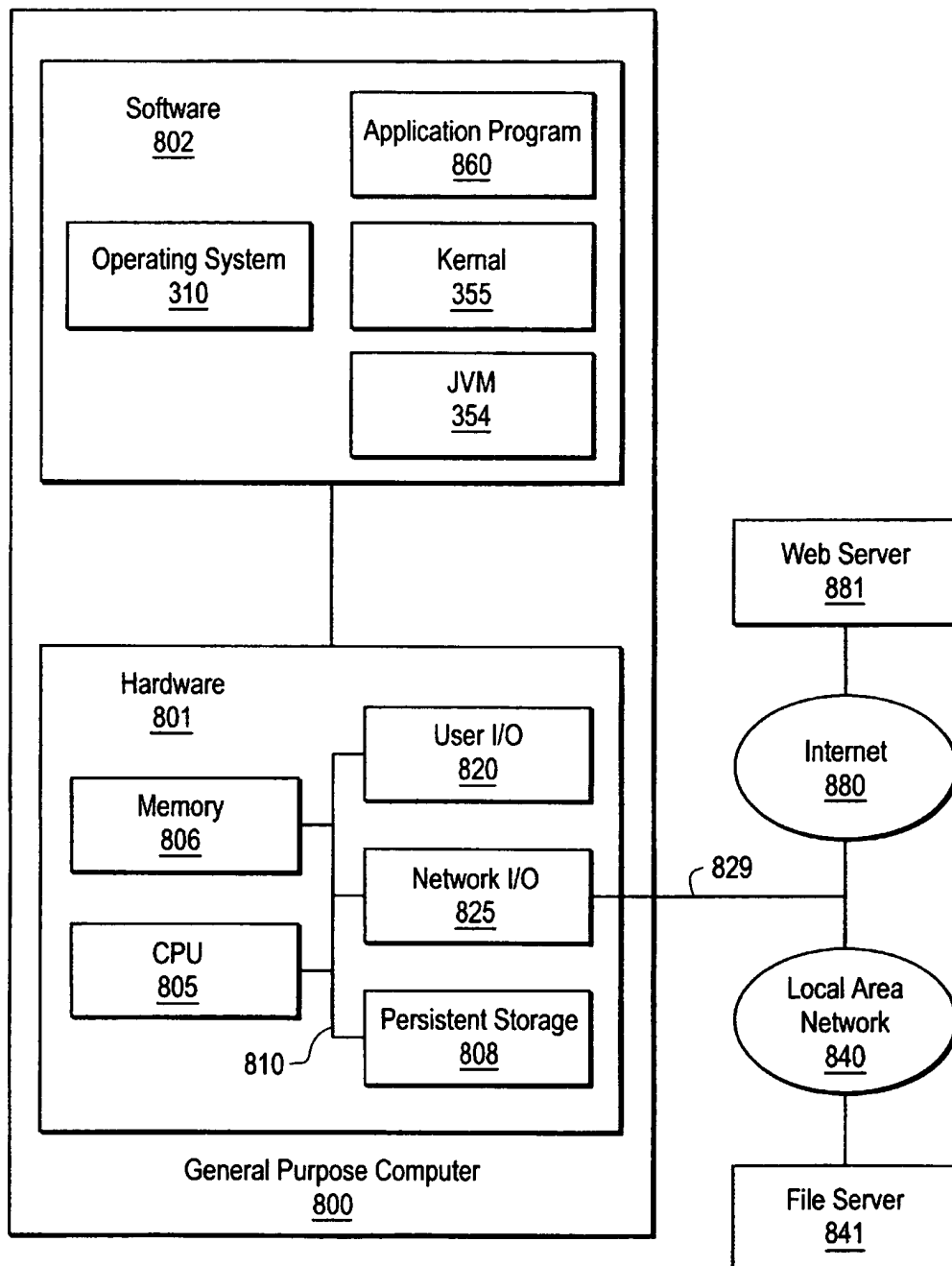


FIG. 8

SMART STUB OR ENTERPRISE JAVA™ BEAN IN A DISTRIBUTED PROCESSING SYSTEM

CROSS REFERENCE TO RELATED APPLICATIONS

This application claims the benefit of U.S. Provisional Application No. 60/107,167, filed Nov. 5, 1998.

The following copending U.S. patent applications are assigned to the assignee of the present application, and their disclosures are incorporated herein by reference:

- (A) Ser. No. 09/405,318 filed Sep. 23, 1999 by Dean B. Jacobs and Anno R. Langen, and originally entitled, "CLUSTERED ENTERPRISE JAVA™ HAVING A MESSAGE PASSING KERNEL IN A DISTRIBUTED PROCESSING SYSTEM";
- (B) Ser. No. 09/405,508 filed Sep. 23, 1999 by Dean B. Jacobs and Eric M. Halpern, and entitled, "A DUPLICATED NAMING SERVICE IN A DISTRIBUTED PROCESSING SYSTEM"; and
- (C) Ser. No. 09/405,500 filed Sep. 23, 1999 by Dean B. Jacobs and Anno R. Langen, and originally entitled, "CLUSTERED ENTERPRISE JAVA™ IN A SECURE DISTRIBUTED PROCESSING SYSTEM".

FIELD OF THE INVENTION

The present invention relates to distributed processing systems and, in particular, computer software in distributed processing systems.

BACKGROUND OF THE INVENTION

There are several types of distributed processing systems. Generally, a distributed processing system includes a plurality of processing devices, such as two computers coupled to a communication medium. Communication mediums may include wired mediums, wireless mediums, or combinations thereof, such as an Ethernet local area network or a cellular network. In a distributed processing system, at least one processing device may transfer information on the communication medium to another processing device.

Client/server architecture 110 illustrated in FIG. 1a is one type of distributed processing system. Client/server architecture 110 includes at least two processing devices, illustrated as client 105 and application server 103. Additional clients may also be coupled to communication medium 104, such as client 108.

Typically, server 103 hosts business logic and/or coordinates transactions in providing a service to another processing device, such as client 103 and/or client 108. Application server 103 is typically programmed with software for providing a service. The software may be programmed using a variety of programming models, such as Enterprise Java™ Bean ("EJB") 100b as illustrated in FIGS. 1a-b. The service may include, for example, retrieving and transferring data from a database, providing an image and/or calculating an equation. For example, server 103 may retrieve data from database 101a in persistent storage 101 over communication medium 102 in response to a request from client 105. Application server 103 then may transfer the requested data over communication medium 104 to client 105.

A client is a processing device which utilizes a service from a server and may request the service. Often a user 106 interacts with client 106 and may cause client 105 to request service over a communication medium 104 from application

server 103. A client often handles direct interactions with end users, such as accepting requests and displaying results.

A variety of different types of software may be used to program application server 103 and/or client 105. One programming language is the Java™ programming language. Java™ application object code is loaded into a Java™ virtual machine ("JVM"). A JVM is a program loaded onto a processing device which emulates a particular machine or processing device. More information on the Java™ programming language may be obtained at <http://www.javasoft.com>, which is incorporated by reference herein.

FIG. 1b illustrates several Java™ Enterprise Application Programming Interfaces ("APIs") 100 that allow Java™ application code to remain independent from underlying transaction systems, data-bases and network infrastructure. Java™ Enterprise APIs 100 include, for example, remote method invocation ("RMI") 100a, EJBs 100b, and Java™ Naming and Directory Interface (JNDI) 100c.

RMI 100a is a distributed programming model often used in peer-to-peer architecture described below. In particular, a set of classes and interfaces enables one Java™ object to call the public method of another Java™ object running on a different JVM.

An instance of EJB 100b is typically used in a client/server architecture described above. An instance of EJB 100b is a software component or a reusable pre-built piece of encapsulated application code that can be combined with other components. Typically, an instance of EJB 100b contains business logic. An EJB 100b instance stored on server 103 typically manages persistence, transactions, concurrency, threading, and security.

JNDI 100c provides directory and naming functions to Java™ software applications.

Client/server architecture 110 has many disadvantages. First, architecture 110 does not scale well because server 103 has to handle many connections. In other words, the number of clients which may be added to server 103 is limited. In addition, adding twice as many processing devices (clients) does not necessarily provide you with twice as much performance. Second, it is difficult to maintain application code on clients 105 and 108. Third, architecture 110 is susceptible to system failures or a single point of failure. If server 103 fails and a backup is not available, client 105 will not be able to obtain the service.

FIG. 1c illustrates a multi-tier architecture 160. Clients 151, 152 manage direct interactions with end users, accepting requests and display results. Application server 153 hosts the application code, coordinates communications, synchronizations, and transactions. Database server 154 and portable storage device 155 provides durable transactional management of the data.

Multi-tier architecture 160 has similar client/server architecture 110 disadvantages described above.

FIG. 2 illustrates peer-to-peer architecture 214. Processing devices 216, 217 and 218 are coupled to communication medium 213. Processing devices 216, 217, and 218 include network software 210a, 210b, and 210c for communicating over medium 213. Typically, each processing device in a peer-to-peer architecture has similar processing capabilities and applications. Examples of peer-to-peer program models include Common Object Request Broker Architecture ("CORBA") and Distributed Object Component Model ("DCOM") architecture.

In a platform specific distributed processing system, each processing device may run the same operating system. This

3

allows the use of proprietary hardware, such as shared disks, multi-tailed disks, and high speed interconnects, for communicating between processing devices. Examples of platform-specific distributed processing systems include IBM® Corporation's S/390® Parallel Sysplex®, Compaq's Tandem Division Himalaya servers, Compaq's Digital Equipment Corporation™ (DEC™) Division OpenVMS™ Cluster software, and Microsoft® Corporation Windows NT® cluster services (Wolfpack).

FIG. 2b illustrates a transaction processing (TP) architecture 220. In particular, TP architecture 220 illustrates a BEA® Systems, Inc. TUXEDO® architecture. TP monitor 224 is coupled to processing devices ATM 221, PC 222, and TP monitor 223 by communication medium 280, 281, and 282, respectively. ATM 221 may be an automated teller machine, PC 222 may be a personal computer, and TP monitor 223 may be another transaction processor monitor. TP monitor 224 is coupled to back-end servers 225, 226, and 227 by communication mediums 283, 284, and 285. Server 225 is coupled to persistent storage device 287, storing database 289, by communication medium 286. TP monitor 224 includes a workflow controller 224a for routing service requests from processing devices, such as ATM 221, PC 222, or TP monitor 223, to various servers such as server 225, 226 and 227. Work flow controller 224a enables (1) workload balancing between servers, (2) limited scalability or allowing for additional servers and/or clients, (3) fault tolerance of redundant backend servers (or a service request may be sent by a workflow controller to a server which has not failed), and (4) session concentration to limit the number of simultaneous connections to back-end servers. Examples of other transaction processing architectures include IBM® Corporation's CICS®, Compaq's Tandem Division Pathway/Ford/TS, Compaq's DEC™ ACMS, and Transarc Corporation's Encina.

TP architecture 220 also has many disadvantages. First, a failure of a single processing device or TP monitor 224 may render the network inoperable. Second, the scalability or number of processing devices (both servers and clients) coupled to TP monitor 224 may be limited by TP monitor 224 hardware and software. Third, flexibility in routing a client request to a server is limited. For example, if communication medium 280 is inoperable, but communication medium 290 becomes available, ATM 221 typically may not request service directly from server 225 over communication medium 290 and must access TP monitor 224. Fourth, a client typically does not know the state of a back-end server or other processing device. Fifth, no industry standard software or APIs are used for load balancing. And sixth, a client typically may not select a particular server even if the client has relevant information which would enable efficient service.

Therefore, it is desirable to provide a distributed processing system and, in particular, distributed processing system software that has the advantages of the prior art distributed processing systems without the inherent disadvantages. The software should allow for industry standard APIs which are typically used in either client/server, multi-tier, or peer-to-peer distributed processing systems. The software should support a variety of computer programming models. Further, the software should enable (1) enhanced fault tolerance, (2) efficient scalability, (3) effective load balancing, and (4) session concentration control. The improved computer software should allow for rerouting or network reconfiguration. Also, the computer software should allow for the determination of the state of a processing device.

SUMMARY OF THE INVENTION

An improved distributed processing system is provided and, in particular, computer software for a distributed pro-

4

cessing system is provided. The computer software improves the fault tolerance of the distributed processing system as well as enables efficient scalability. The computer software allows for efficient load balancing and session concentration. The computer software supports rerouting or reconfiguration of a computer network. The computer software supports a variety of computer programming models and allows for the use of industry standard APIs that are used in both client/server and peer-to-peer distributed processing architectures. The computer software enables a determination of the state of a server or other processing device. The computer software also supports message forwarding under a variety of circumstances, including a security model.

According to one aspect of the present invention, a distributed processing system comprises a communication medium coupled to a first processing device and a second processing device. The first processing device includes a first software program emulating a processing device ("JVM1") including a first kernel software layer having a data structure ("RJVM1"). The second processing device includes a first software program emulating a processing device ("JVM2") including a first kernel software layer having a data structure ("RJVM2"). A message from the first processing device is transferred to the second processing device through the first kernel software layer and the first software program in the first processing device to the first kernel software layer and the first software program in the second processing device.

According to another aspect of the present invention, the first software program in the first processing device is a Java™ virtual machine ("JVM") and the data structure in the first processing device is a remote Java™ virtual machine ("RJVM"). Similarly, the first software program in the second processing device is a JVM and the data structure in the second processing device is a RJVM. The RJVM in the second processing device corresponds to the JVM in the first processing device.

According to another aspect of the present invention, the RJVM in the first processing device includes a socket manager software component, a thread manager software component, a message routing software component, a message compression software component, and/or a peer-gone detection software component.

According to another aspect of the present invention, the first processing device communicates with the second processing device using a protocol selected from the group consisting of Transmission Control Protocol ("TCP"), Secure Sockets Layer ("SSL"), Hypertext Transport Protocol ("HTTP") tunneling, and Internet InterORB Protocol ("IIOP") tunneling.

According to another aspect of the present invention, the first processing device includes memory storage for a Java™ application.

According to another aspect of the present invention, the first processing device is a peer of the second processing device. Also, the first processing device is a server and the second processing device is a client.

According to another aspect of the present invention, a second communication medium is coupled to the second processing device. A third processing device is coupled to the second communication medium. The third processing device includes a first software program emulating a processing device ("JVM3"), including a kernel software layer having a first data structure ("RJVM1"), and a second data structure ("RJVM2").

According to still another aspect of the present invention, the first processing device includes a stub having a replica-

5

handler software component. The replica-handler software component includes a load balancing software component and a failover software component.

According to another aspect of the present invention, the first processing device includes an Enterprise Java™ Bean object.

According to still another aspect of the present invention, the first processing device includes a naming tree having a pool of stubs stored at a node of the tree and the second processing device includes a duplicate of the naming tree.

According to still another aspect of the present invention, the first processing device includes an application program coded in a stateless program model and the application program includes a stateless session bean.

According to still another aspect of the present invention, the first processing device includes an application program coded in a stateless factory program model and the application program includes a stateful session bean.

According to still another aspect of the present invention, the first processing device includes an application program coded in a stateful program model and the application program includes an entity session bean.

According to still another aspect of the present invention, an article of manufacture including an information storage medium is provided. The article of manufacture comprises a first set of digital information for transferring a message from a RJVM in a first processing device to a RJVM in a second processing device.

According to another aspect of the present invention, the article of manufacture comprises a first set of digital information, including a stub having a load balancing software program for selecting a service provider from a plurality of service providers.

According to another aspect of the present invention, the stub has a failover software component for removing a failed service provider from the plurality of service providers.

According to another aspect of the present invention, the load balancing software component selects a service provider based on an affinity for a particular service provider.

According to another aspect of the present invention, the load balancing software component selects a service provider in a round robin manner.

According to another aspect of the present invention, the load balancing software component randomly selects a service provider.

According to another aspect of the present invention, the load balancing software component selects a service provider from the plurality of service providers based upon the load of each service provider.

According to another aspect of the present invention, the load balancing software component selects a service provider from the plurality of service providers based upon the data type requested.

According to another aspect of the present invention, the load balancing software component selects a service provider from the plurality of service providers based upon the closest physical service provider.

According to another aspect of the present invention, the load balancing software component selects a service provider from the plurality of service providers based upon a time period in which each service provider responds.

According to another aspect of the present invention, the article of manufacture comprises a first set of digital information, including an Enterprise Java™ Bean object for selecting a service provider from a plurality of service providers.

6

According to another aspect of the present invention, a stub is stored in a processing device in a distributed processing system. The stub includes a method comprising the steps of obtaining a list of service providers and selecting a service provider from the list of service providers.

According to another aspect of the present invention, the method further includes removing a failed service provider from the list of service providers.

According to still another aspect of the present invention, an apparatus comprises a communication medium coupled to a first processing device and a second processing device. The first processing device stores a naming tree including a remote method invocation ("RMI") stub for accessing a service provider. The second processing device has a duplicate naming tree and the service provider.

According to another aspect of the present invention, the naming tree has a node including a service pool of current service providers.

According to another aspect of the present invention, the service pool includes a stub.

According to another aspect of the present invention, a distributed processing system comprises a first computer coupled to a second computer. The first computer has a naming tree, including a remote invocation stub for accessing a service provider. The second computer has a replicated naming tree and the service provider.

According to another aspect of the present invention, a distributed processing system comprising a first processing device coupled to a second processing device is provided. The first processing device has a JVM and a first kernel software layer including a first RJVM. The second processing device includes a first JVM and a first kernel software layer including a second RJVM. A message may be transferred from the first processing device to the second processing device when there is not a socket available between the first JVM and the second JVM.

According to another aspect of the present invention, the first processing device is running under an applet security model, behind a firewall or is a client, and the second processing device is also a client.

Other aspects and advantages of the present invention can be seen upon review of the figures, the detailed description, and the claims which follow.

BRIEF DESCRIPTION OF THE FIGURES

FIG. 1a illustrates a prior art client/server architecture;

FIG. 1b illustrates a prior art Java™ enterprise APIs;

FIG. 1c illustrates a multi-tier architecture;

FIG. 2a illustrates a prior art peer-to-peer architecture;

FIG. 2b illustrates a prior art transaction processing architecture;

FIG. 3a illustrates a simplified software block diagram of an embodiment of the present invention;

FIG. 3b illustrates a simplified software block diagram of the kernel illustrated in FIG. 3a;

FIG. 3c illustrates a clustered enterprise Java™ architecture;

FIG. 4 illustrates a clustered enterprise Java™ naming service architecture;

FIG. 5a illustrates a smart stub architecture;

FIG. 5b illustrates an EJB object architecture;

FIG. 6a is a control flow chart illustrating a load balancing method;

7

FIGS. 6b-g are control flow charts illustrating load balancing methods;

FIG. 7 is a control flow chart illustrating a failover method;

FIG. 8 illustrates hardware and software components of a client/server in the clustered enterprise Java™ architecture shown in FIGS. 3-5.

The invention will be better understood with reference to the drawings and detailed description below. In the drawings, like reference numerals indicate like components.

DETAILED DESCRIPTION

I. Clustered Enterprise Java™ Distributed Processing System

A. Clustered Enterprise Java™ Software Architecture

FIG. 3a illustrates a simplified block diagram 380 of the software layers in a processing device of a clustered enterprise Java™ system, according to an embodiment of the present invention. A detailed description of a clustered enterprise Java™ distributed processing system is described below. The first layer of software includes a communication medium software driver 351 for transferring and receiving information on a communication medium, such as an ethernet local area network. An operating system 310 including a transmission control protocol ("TCP") software component 353 and internet protocol ("IP") software component 352 are upper software layers for retrieving and sending packages or blocks of information in a particular format. An "upper" software layer is generally defined as a software component which utilizes or accesses one or more "lower" software layers or software components. A JVM 354 is then implemented. A kernel 355 having a remote Java™ virtual machine 356 is then layered on JVM 354. Kernel 355, described in detail below, is used to transfer messages between processing devices in a clustered enterprise Java™ distributed processing system. Remote method invocation 357 and enterprise Java™ bean 358 are upper software layers of kernel 355. EJB 358 is a container for a variety of Java™ applications.

FIG. 3b illustrates a detailed view of kernel 355 illustrated in FIG. 3a. Kernel 355 includes a socket manager component 363, thread manager 364 component, and RJVM 356. RJVM 356 is a data structure including message routing software component 360, message compression software component 361 including abbreviation table 161c, and peer-gone detection software component 362. RJVM 356 and thread manager component 364 interact with socket manager component 363 to transfer information between processing devices.

B. Distributed Processing System

FIG. 3 illustrates a simplified block diagram of a clustered enterprise Java™ distributed processing system 300. Processing devices are coupled to communication medium 301. Communication medium 301 may be a wired and/or wireless communication medium or combination thereof. In an embodiment, communication medium 301 is a local-area-network (LAN). In an alternate embodiment, communication medium 301 is a world-area-network (WAN) such as the Internet or World Wide Web. In still another embodiment, communication medium 301 is both a LAN and a WAN.

A variety of different types of processing devices may be coupled to communication medium 301. In an embodiment, a processing device may be a general purpose computer 100 as illustrated in FIG. 8 and described below. One of ordinary skill in the art would understand that FIG. 8 and the below

8

description describes one particular type of processing device where multiple other types of processing devices with a different software and hardware configurations could be utilized in accordance with an embodiment of the present invention. In an alternate embodiment, a processing device may be a printer, handheld computer, laptop computer, scanner, cellular telephone, pager, or equivalent thereof.

FIG. 3c illustrates an embodiment of the present invention in which servers 302 and 303 are coupled to communication medium 301. Server 303 is also coupled to communication medium 305 which may have similar embodiments as described above in regard to communication medium 301. Client 304 is also coupled to communication medium 305. In an alternate embodiment, client 304 may be coupled to communication medium 301 as illustrated by the dashed line and box in FIG. 3c. It should be understood that in alternate embodiments, server 302 is (1) both a client and a server, or (2) a client. Similarly, FIG. 3 illustrates an embodiment in which three processing devices are shown wherein other embodiments of the present invention include multiple other processing devices or communication mediums as illustrated by the ellipses.

Server 302 transfers information over communication medium 301 to server 303 by using network software 302a and network software 303a, respectively. In an embodiment, network software 302a, 303a, and 304a include communication medium software driver 351, Transmission Control Protocol software 353, and Internet Protocol software 352 ("TCP/IP"). Client 304 also includes network software 304a for transferring information to server 303 over communication medium 305. Network software 303a in server 303 is also used to transfer information to client 304 by way of communication medium 305.

According to an embodiment of the present invention, each processing device in clustered enterprise Java™ architecture 300 includes a message-passing kernel 355 that supports both multi-tier and peer-to-peer functionality. A kernel is a software program used to provide fundamental services to other software programs on a processing device.

In particular, server 302, server 303, and client 304 have kernels 302b, 303b, and 304b, respectively. In particular, in order for two JVMs to interact, whether they are clients or servers, each JVM constructs an RJVM representing the other. Messages are sent from the upper layer on one side, through a corresponding RJVM, across the communication medium, through the peer RJVM, and delivered to the upper layer on the other side. In various embodiments, messages can be transferred using a variety of different protocols, including, but not limited to, Transmission Control Protocol/Internet Protocol ("TCP/IP"), Secure Sockets Layer ("SSL"), Hypertext Transport Protocol ("HTTP") tunneling, and Internet InterORB Protocol ("IIOP") tunneling, and combinations thereof. The RJVMs and socket managers create and maintain the sockets underlying these protocols and share them between all objects in the upper layers. A socket is a logical location representing a terminal between processing devices in a distributed processing system. The kernel maintains a pool of execute threads and thread manager software component 364 multiplexes the threads between socket reading and request execution. A thread is a sequence of executing program code segments or functions.

For example, server 302 includes JVM1 and Java™ application 302c. Server 302 also includes a RJVM2 representing the JVM2 of server 303. If a message is to be sent from server 302 to server 303, the message is sent through RJVM2 in server 302 to RJVM1 in server 303.

C. Message Forwarding

Clustered enterprise Java™ network 300 is able to forward a message through an intermediate server. This functionality is important if a client requests a service from a back-end server through a front-end gateway. For example, a message from server 302 (client 302) and, in particular, JVM1 may be forwarded to client 304 (back-end server 304) or JVM3 through server 303 (front-end gateway) or JVM2. This functionality is important in controlling session concentration or how many connections are established between a server and various clients.

Further, message forwarding may be used in circumstances where a socket cannot be created between two JVMs. For example, a sender of a message is running under the applet security model which does not allow for a socket to be created to the original server. A detailed description of the applet security model is provided at <http://www.javasoft.com>, which is incorporated herein by reference. Another example includes when the receiver of the message is behind a firewall. Also, as described below, message forwarding is applicable if the sender is a client and the receiver is a client and thus does not accept incoming sockets.

For example, if a message is sent from server 302 to client 304, the message would have to be routed through server 303. In particular, a message handoff, as illustrated by 302f, between RJVM3 (representing client 304) would be made to RJVM2 (representing server 303) in server 302. The message would be transferred using sockets 302e between RJVM2 in server 302 and RJVM1 in server 303. The message would then be handed off, as illustrated by dashed line 303f, from RJVM1 to RJVM3 in server 303. The message would then be passed between sockets of RJVM3 in server 303 and RJVM2 in client 304. The message then would be passed, as illustrated by the dashed line 304f, from RJVM2 in client 304 to RJVM1 in client 304.

D. Rerouting

An RJVM in client/server is able to switch communication paths or communication mediums to other RJVMs at any time. For example, if client 304 creates a direct socket to server 302, server 302 is able to start using the socket instead of message forwarding through server 303. This embodiment is illustrated by a dashed line and box representing client 304. In an embodiment, the use of transferring messages by RJVMs ensures reliable, in-order message delivery after the occurrence of a network reconfiguration. For example, if client 304 was reconfigured to communication medium 301 instead of communication medium 305 as illustrated in FIG. 3. In an alternate embodiment, messages may not be delivered in order.

An RJVM performs several end-to-end operations that are carried through routing. First, an RJVM is responsible for detecting when a respective client/server has unexpectedly died. In an embodiment, peer-gone selection software component 362, as illustrated in FIG. 3b, is responsible for this function. In an embodiment, an RJVM sends a heartbeat message to other clients/servers when no other message has been sent in a predetermined time period. If the client/server does not receive a heartbeat message in the predetermined count time, a failed client/server which should have sent the heartbeat, is detected. In an embodiment, a failed client/server is detected by connection timeouts or if no messages have been sent by the failed client/server in a predetermined amount of time. In still another embodiment, a failed socket indicates a failed server/client.

Second, during message serialization, RJVMs, in particular, message compression software 360, abbreviate

commonly transmitted data values to reduce message size. To accomplish this, each JVM/RJVM pair maintains matching abbreviation tables. For example, JVM1 includes an abbreviation table and RJVM1 includes a matching abbreviation table. During message forwarding between an intermediate server, the body of a message is not deserialized on the intermediate server in route.

E. Multi-tier/Peer-to-Peer Functionality

Clustered enterprise Java™ architecture 300 allows for multitier and peer-to-peer programming.

Clustered enterprise Java™ architecture 300 supports an explicit syntax for client/server programming consistent with a multitier distributed processing architecture. As an example, the following client-side code fragment writes an informational message to a server's log file:

```
T3Client clnt=new T3Client("t3://acme:7001");
LogServices log=clnt.getT3Services().log();
log.info("Hello from a client");
```

The first line establishes a session with the acme server using the t3 protocol. If RJVMs do not already exist, each JVM constructs an RJVM for the other and an underlying TCP socket is established. The client-side representation of this session—the T3Client object—and the server-side representation communicate through these RJVMs. The server-side supports a variety of services, including database access, remote file access, workspaces, events, and logging. The second line obtains a LogServices object and the third line writes the message.

Clustered enterprise Java™ computer architecture 300 also supports a server-neutral syntax consistent with a peer-to-peer distributed processing architecture. As an example, the following code fragment obtains a stub for an RMI object from the JNDI-compliant naming service on a server and invokes one of its methods.

```
Hashtable env=new Hashtable();
env.put(Context.PROVIDER_URL, "t3://acme:7001");
env.put(Context.INITIAL_CONTEXT_FACTORY,
"weblogic.jndi.WebLogicInitialContextFactory");
Context ctx=new InitialContext(env);
Example e=(Example) ctx.lookup("acme.eng.example");
result=e.example(37);
```

In an embodiment, JNDI naming contexts are packaged as RMI objects to implement remote access. Thus, the above code illustrates a kind of RMI bootstrapping. The first four lines obtain an RMI stub for the initial context on the acme server. If RJVMs do not already exist, each side constructs an RJVM for the other and an underlying TCP socket for the t3 protocol is established. The caller-side object—the RMI stub—and the callee-side object—an RMI impl—communicate through the RJVMs. The fifth line looks up another RMI object, an Example, at the name acme.eng.example and the sixth line invokes one of the Example methods. In an embodiment, the Example impl is not on the same processing device as the naming service. In another embodiment, the Example impl is on a client. Invocation of the Example object leads to the creation of the appropriate RJVMs if they do not already exist.

II. Replica-Aware or Smart Stubs/EJB Objects

In FIG. 3c, a processing device is able to provide a service to other processing devices in architecture 300 by replicating RMI and/or EJB objects. Thus, architecture 300 is easily scalable and fault tolerant. An additional service may easily be added to architecture 300 by adding replicated RMI and/or EJB objects to an existing processing device or newly added processing device. Moreover, because the RMI and/or

11

EJB objects can be replicated throughout architecture 300, a single processing device, multiple processing devices, and/or a communication medium may fail and still not render architecture 300 inoperable or significantly degraded.

FIG. 5a illustrates a replica-aware ("RA") or Smart stub 580 in architecture 500. Architecture 500 includes client 504 coupled to communication medium 501. Servers 502 and 503 are coupled to communication medium 501, respectively. Persistent storage device 509 is coupled to server 502 and 503 by communication medium 560 and 561, respectively. In various embodiments, communication medium 501, 560, and 561 may be wired and/or wireless communication mediums as described above. Similarly, in an embodiment, client 504, server 502, and server 503 may be both clients and servers as described above. One of ordinary skill in the art would understand that in alternate embodiments, multiple other servers and clients may be included in architecture 500 as illustrated by ellipses. Also, as stated above, in alternate embodiments, the hardware and software configuration of client 504, server 502 and server 503 is described below and illustrated in FIG. 8.

RA RMI stub 580 is a Smart stub which is able to find out about all of the service providers and switch between them based on a load balancing method 507 and/or failover method 508. In an embodiment, an RA stub 580 includes a replica handler 506 that selects an appropriate load balancing method 507 and/or failover method 507. In an alternate embodiment, a single load balancing method and/or single failover method is implemented. In alternate embodiments, replica handler 506 may include multiple load balancing methods and/or multiple failover methods and combinations thereof. In an embodiment, a replica handler 506 implements the following interface:

```
public interface ReplicaHandler {
    Object loadBalance(Object currentProvider) throws
    RefreshAbortedException;
    Object failOver(Object failedProvider,
    RemoteException e) throws
    RemoteException;
}
```

Immediately before invoking a method, RA stub 580 calls load balance method 507, which takes the current server and returns a replacement. For example, client 504 may be using server 502 for retrieving data for database 509a or personal storage device 509. Load balance method 507 may switch to server 503 because server 502 is overloaded with service requests. Handler 506 may choose a server replacement entirely on the caller, perhaps using information about server 502 load, or handler 506 may request server 502 for retrieving a particular type of data. For example, handler 506 may select a particular server for calculating an equation because the server has enhanced calculation capability. In an embodiment, replica handler 506 need not actually switch providers on every invocation because replica handler 506 is trying to minimize the number of connections that are created.

FIG. 6a is a control flow diagram illustrating the load balancing software 507 illustrated in FIGS. 5a-b. It should be understood that FIG. 6a is a control flow diagram illustrating the logical sequence of functions or steps which are completed by software in load balancing method 507. In alternate embodiments, additional functions or steps are completed. Further, in an alternate embodiment, hardware may perform a particular function or all the functions.

Load balancing software 507 begins as indicated by circle 600. A determination is then made in logic block 601 as to whether the calling thread established "an affinity" for a

12

particular server. A client has an affinity for the server that coordinates its current transaction and a server has an affinity for itself. If an affinity is established, control is passed to logic block 602, otherwise control is passed to logic block 604. A determination is made in logic block 602 whether the affinity server provides the service requested. If so, control is passed to logic block 603. Otherwise, control is passed to logic block 604. The provider of the service on the affinity server is returned to the client in logic block 603. In logic block 604, a naming service is contacted and an updated list of the current service providers is obtained. A getNextProvider method is called to obtain a service provider in logic block 605. Various embodiments of the getNextProvider method are illustrated in FIGS. 6b-g and described in detail below. The service is obtained in logic block 606. Failover method 508 is then called if service is not provided in logic block 606 and load balancing method 507 exits as illustrated by logic block 608. An embodiment of failover method 508 is illustrated in FIG. 7 and described in detail below.

FIGS. 6b-g illustrate various embodiments of a getNextProvider method used in logic block 605 of FIG. 6a. As illustrated in FIG. 6b, the getNextProvider method selects a service provider in a round robin manner. A getNextProvider method 620 is entered as illustrated by circle 621. A list of current service providers is obtained in logic block 622. A pointer is incremented in logic block 623. The next service provider is selected based upon the pointer in logic block 624 and the new service provider is returned in logic block 625 and getNextProvider method 620 exits as illustrated by circle 626.

FIG. 6c illustrates an alternate embodiment of a getNextProvider method which obtains a service provider by selecting a service provider randomly. A getNextProvider method 630 is entered as illustrated by circle 631. A list of current service providers is obtained as illustrated by logic block 632. The next service provider is selected randomly as illustrated by logic block 633 and a new service provider is returned in logic block 634. The getNextProvider method 630 then exits, as illustrated by circle 635.

Still another embodiment of a getNextProvider method is illustrated in FIG. 6d which obtains a service provider based upon the load of the service providers. A getNextProvider method 640 is entered as illustrated by circle 641. A list of current service providers is obtained in logic block 642. The load of each service provider is obtained in logic block 643. The service provider with the least load is then selected in logic block 644. The new service provider is then returned in logic block 645 and getNextProvider method 640 exits as illustrated by circle 646.

An alternate embodiment of a getNextProvider method is illustrated in FIG. 6e which obtains a service provider based upon the type of data obtained from the service provider. A getNextProvider method 650 is entered as illustrated by circle 651. A list of current service providers is obtained in logic block 652. The type of data requested from the service providers is determined in logic block 653. The service provider is then selected based on the data type in logic block 654. The service provider is returned in logic block 655 and getNextProvider method 650 exits as illustrated by circle 656.

Still another embodiment of a getNextProvider method is illustrated in FIG. 6f which selects a service provider based upon the physical location of the service providers. A getNextProvider method 660 is entered as illustrated by circle 661. A list of service providers is obtained as illustrated by logic block 662. The physical distance to each service provider is determined in logic block 663 and the

13

service provider which has the closest physical distance to the requesting client is selected in logic block 664. The new service provider is then returned in logic block 665 and the getNextProvider method 660 exits as illustrated by circle 666.

Still a further embodiment of the getNextProvider method is illustrated in FIG. 6g and selects a service provider based on the amount of time taken for the service provider to respond to previous requests. Control of getNextProvider method 670 is entered as illustrated by circle 671. A list of current service providers is obtained in logic block 672. The time period for each service provider to respond to a particular message is determined in logic block 673. The service provider which responds in the shortest time period is selected in logic block 674. The new service provider is then returned in logic block 675 and control from getNextProvider method 670 exits as illustrated by circle 676.

If invocation of a service method fails in such a way that a retry is warranted, RA stub 580 calls failover method 508, which takes the failed server and an exception indicating what the failure was and returns a new server for the retry. If a new server is unavailable, RA stub 580 throws an exception.

FIG. 7 is a control flow chart illustrating failover software 508 shown in FIGS. 5a-b. Failover method 508 is entered as illustrated by circle 700. A failed provider from the list of current providers of services is removed in logic block 701. A getNextProvider method is then called in order to obtain a service provider. The new service provider is then returned in logic block 703 and failover method 508 exits as illustrated by circle 704.

While FIGS. 6-7 illustrate embodiments of a replica handler 506, alternate embodiments include the following functions or combinations thereof implemented in a round robin manner.

First, a list of servers or service providers of a service is maintained. Whenever the list needs to be used and the list has not been recently updated, handler 506 contacts a naming service as described below and obtains an up-to-date list of providers.

Second, if handler 506 is about to select a provider from the list and there is an existing RJVM-level connection to the hosting server over which no messages have been received during the last heartbeat period, handler 506 skips that provider. In an embodiment, a server may later recover since death of peer is determined after several such heartbeat periods. Thus, load balancing on the basis of server load is obtained.

Third, when a provider fails, handler 506 removes the provider from the list. This avoids delays caused by repeated attempts to use non-working service providers.

Fourth, if a service is being invoked from a server that hosts a provider of the service, then that provider is used. This facilitates co-location of providers for chained invokes of services.

Fifth, if a service is being invoked within the scope of a transaction and the server acting as transaction coordinator hosts a provider of the service, then that provider is used. This facilitates co-location of providers within a transaction.

The failures that can occur during a method invocation may be classified as being either (1) application-related, or (2) infrastructure-related. RA stub 580 will not retry an operation in the event of an application-related failure, since there can be no expectation that matters will improve. In the event of an infrastructure-related failure, RA stub 580 may or may not be able to safely retry the operation. Some initial non-idempotent operation, such as incrementing the value of

14

a field in a database, might have completed. In an embodiment, RA stub 580 will retry after an infrastructure failure only if either (1) the user has declared that the service methods are idempotent, or (2) the system can determine that processing of the request never started. As an example of the latter, RA stub 580 will retry if, as part of load balancing method, stub 580 switches to a service provider whose host has failed. As another example, a RA stub 580 will retry if it gets a negative acknowledgment to a transactional operation.

A RMI compiler recognizes a special flag that instructs the compiler to generate an RA stub for an object. An additional flag can be used to specify that the service methods are idempotent. In an embodiment, RA stub 580 will use the replica handler described above and illustrated in FIG. 5a. An additional flag may be used to specify a different handler. In addition, at the point a service is deployed, i.e., bound into a clustered naming service as described below, the handler may be overridden.

FIG. 5b illustrates another embodiment of the present invention in which an EJB object 551 is used instead of a stub, as shown in FIG. 5a.

III. Replicated JNDI-compliant Naming Service

As illustrated in FIG. 4, access to service providers in architecture 400 is obtained through a JNDI-compliant naming service, which is replicated across architecture 400 so there is no single point of failure. Accordingly, if a processing device which offers a JNDI-compliant naming service fails, another processing device having a replicated naming service is available. To offer an instance of a service, a server advertises a provider of the service at a particular node in a replicated naming tree. In an embodiment, each server adds a RA stub for the provider to a compatible service pool stored at the node in the server's copy of the naming tree. If the type of a new offer is incompatible with the type of offers in an existing pool, the new offer is made pending and a callback is made through a ConflictHandler interface. After either type of offer is retracted, the other will ultimately be installed everywhere. When a client looks up the service, the client obtains a RA stub that contacts the service pool to refresh the client's list of service providers.

FIG. 4 illustrates a replicated naming service in architecture 400. In an embodiment, servers 302 and 303 offer an example service provider P1 and P2, respectively, and has a replica of the naming service tree 402 and 403, respectively. The node acme.eng.example in naming service tree 402 and 403 has a service pool 402a and 403a, respectively, containing a reference to Example service provider P1 and P2. Client 304 obtains a RA stub 304e by doing a naming service lookup at the acme.eng.example node. Stub 304e contacts an instance of a service pool to obtain a current list of references to available service providers. Stub 304e may switch between the instances of a service pool as needed for load-balancing and failover.

Stubs for the initial context of the naming service are replica-aware or Smart stubs which initially load balance among naming service providers and switch in the event of a failure. Each instance of the naming service tree contains a complete list of the current naming service providers. The stub obtains a fresh list from the instance it is currently using. To bootstrap this process, the system uses Domain Naming Service ("DNS") to find a (potentially incomplete) initial list of instances and obtains the complete list from one of them. As an example, a stub for the initial context of the naming service can be obtained as follows:

```
Hashtable env=new Hashtable( );
env.put(Context.PROVIDER_URL, "t3://
acmeCluster:7001");
```


15

```
env.put(Context.INITIAL_CONTEXT_FACTORY,
"weblogic.jndi.WebLogicInitialContextFactor");
```

```
Context ctx=new InitialContext(env);
```

Some subset of the servers in an architecture have been bound into DNS under the name acmeCluster. Moreover, an application is still able to specify the address of an individual server, but the application will then have a single point of failure when the application first attempts to obtain a stub.

A reliable multicast protocol is desirable. In an embodiment, provider stubs are distributed and replicated naming trees are created by an IP multicast or point-to-point protocol. In an IP multicast embodiment, there are three kinds of messages: Heartbeats, Announcements, and StateDumps. Heartbeats are used to carry information between servers and, by their absence, to identify failed servers. An Announcement contains a set of offers and retractions of services. The Announcements from each server are sequentially numbered. Each receiver processes an Announcement in order to identify lost Announcements. Each server includes in its Heartbeats the sequence number of the last Announcement it has sent. Negative Acknowledgments ("NAKs") for a lost Announcement are included in subsequent outgoing Heartbeats. To process NAKs, each server keeps a list of the last several Announcements that the server has sent. If a NAK arrives for an Announcement that has been deleted, the server sends a StateDump, which contains a complete list of the server's services and the sequence number of its next Announcement. When a new server joins an existing architecture, the new server NAKs for the first message from each other server, which results in StateDumps being sent. If a server does not receive a Heartbeat from another server after a predetermined period of time, the server retracts all services offered by the server not generating a Heartbeat.

IV. Programming Models

Applications used in the architecture illustrated in FIGS. 3-5 use one of three basic programming models: (1) stateless or direct, (2) stateless factory or indirect, or (3) stateful or targeted, depending on the way the application state is to be treated. In the stateless model, a Smart stub returned by a naming-service lookup directly references service providers.

```
Example e=(Example) ctx.lookup("acme.eng.example");
result1=e.example(37);
result2=e.example(38);
```

In this example, the two calls to example may be handled by different service providers since the Smart stub is able to switch between them in the interests of load balancing. Thus, the Example service object cannot internally store information on behalf of the application. Typically the stateless model is used only if the provider is stateless. As an example, a pure stateless provider might compute some mathematical function of its arguments and return the result. Stateless providers may store information on their own behalf, such as for accounting purposes. More importantly, stateless providers may access an underlying persistent storage device and load application state into memory on an as-needed basis. For example, in order for example to return the running sum of all values passed to it as arguments, example might read the previous sum from a database, add in its current argument, write the new value out, and then return it. This stateless service model promotes scalability.

In the stateless factory programming model, the Smart stub returned by the lookup is a factory that creates the desired service providers, which are not themselves Smart stubs.

16

```
ExampleFactory gf=(ExampleFactory)
```

```
ctx.lookup("acme.eng.example");
```

```
Example e=gf.create( );
```

```
result1=e.example(37);
```

```
result2=e.example(38);
```

In this example, the two calls to example are guaranteed to be handled by the same service provider. The service provider may therefore safely store information on behalf of the application. The stateless factory model should be used when the caller needs to engage in a "conversation" with the provider. For example, the caller and the provider might engage in a back-and-forth negotiation. Replica-aware stubs are generally the same in the stateless and stateless factory models, the only difference is whether the stubs refer to service providers or service provider factories.

A provider factory stub may failover at will in its effort to create a provider, since this operation is idempotent. To further increase the availability of an indirect service, application code must contain an explicit retry loop around the service creation and invocation.

```
while (true) {
    try {
        Example e=gf.create( );
        result1=e.example(37);
        result2=e.example(38);
        break;
    } catch (Exception e) {
        if (!retryWarranted(e))
            throw e;
    }
}
```

This would, for example, handle the failure of a provider e that was successfully created by the factory. In this case, application code should determine whether non-idempotent operations completed. To further increase availability, application code might attempt to undo such operations and retry.

In the stateful programming model, a service provider is a long-lived, stateful object identified by some unique system-wide key. Examples of "entities" that might be accessed using this model include remote file systems and rows in a database table. A targeted provider may be accessed many times by many clients, unlike the other two models where each provider is used once by one client. Stubs for targeted providers can be obtained either by direct lookup, where the key is simply the naming-service name, or through a factory, where the key includes arguments to the create operation. In either case, the stub will not do load balancing or failover. Retries, if any, must explicitly obtain the stub again.

There are three kinds of beans in EJB, each of which maps to one of the three programming models. Stateless session beans are created on behalf of a particular caller, but maintain no internal state between calls. Stateless session beans map to the stateless model. Stateful session beans are created on behalf of a particular caller and maintain internal state between calls. Stateful session beans map to the stateless factory model. Entity beans are singular, stateful objects identified by a system-wide key. Entity beans map to the stateful model. All three types of beans are created by a factory called an EJB home. In an embodiment, both EJB homes and the beans they create are referenced using RMI. In an architecture as illustrated in FIGS. 3-5, stubs for an EJB home are Smart stubs. Stubs for stateless session beans are Smart stubs, while stubs for stateful session beans and entity beans are not. The replica handler to use for an EJB-based service can be specified in its deployment descriptor.

17

To create an indirect RMI-based service, which is required if the object is to maintain state on behalf of the caller, the application code must explicitly construct the factory. A targeted RMI-based service can be created by running the RMI compiler without any special flags and then binding the resulting service into the replicated naming tree. A stub for the object will be bound directly into each instance of the naming tree and no service pool will be created. This provides a targeted service where the key is the naming-service name. In an embodiment, this is used to create remote file systems.

V. Hardware and Software Components

FIG. 8 shows hardware and software components of an exemplary server and/or client as illustrated in FIGS. 3-5. The system of FIG. 8 includes a general-purpose computer 800 connected by one or more communication mediums, such as connection 829, to a LAN 840 and also to a WAN, here illustrated as the Internet 880. Through LAN 840, computer 800 can communicate with other local computers, such as a file server 841. In an embodiment, file server 801 is server 303 as illustrated in FIG. 3. Through the Internet 880, computer 800 can communicate with other computers, both local and remote, such as World Wide Web server 881. In an embodiment, Web server 881 is server 303 as illustrated in FIG. 3. As will be appreciated, the connection from computer 800 to Internet 880 can be made in various ways, e.g., directly via connection 829, or through local-area network 840, or by modem (not shown).

Computer 800 is a personal or office computer that can be, for example, a workstation, personal computer, or other single-user or multi-user computer system; an exemplary embodiment uses a Sun SPARC-20 workstation (Sun Microsystems, Inc., Mountain View, Calif.). For purposes of exposition, computer 800 can be conveniently divided into hardware components 801 and software components 802; however, persons of ordinary skill in the art will appreciate that this division is conceptual and somewhat arbitrary, and that the line between hardware and software is not a hard and fast one. Further, it will be appreciated that the line between a host computer and its attached peripherals is not a hard and fast one, and that in particular, components that are considered peripherals of some computers are considered integral parts of other computers. Thus, for example, user I/O 820 can include a keyboard, a mouse, and a display monitor, each of which can be considered either a peripheral device or part of the computer itself, and can further include a local printer, which is typically considered to be a peripheral. As another example, persistent storage 808 can include a CD-ROM (compact disc read-only memory) unit, which can be either peripheral or built into the computer.

Hardware components 801 include a processor (CPU) 805, memory 806, persistent storage 808, user I/O 820, and network interface 825 which are coupled to bus 810. These components are well understood by those of skill in the art and, accordingly, need be explained only briefly here.

Processor 805 can be, for example, a microprocessor or a collection of microprocessors configured for multiprocessing.

Memory 806 can include read-only memory (ROM), randomaccess memory (RAM), virtual memory, or other memory technologies, singly or in combination. Persistent storage 808 can include, for example, a magnetic hard disk, a floppy disk, or other persistent read-write data storage technologies, singly or in combination. It can further include mass or archival storage, such as can be provided by CD-ROM or other large-capacity storage technology. (Note that file server 841 provides additional storage capability that processor 805 can use.)

18

User I/O (input/output) hardware 820 typically includes a visual display monitor such as a CRT or flat-panel display, an alphanumeric keyboard, and a mouse or other pointing device, and optionally can further include a printer, an optical scanner, or other devices for user input and output.

Network I/O hardware 825 provides an interface between computer 800 and the outside world. More specifically, network I/O 825 lets processor 805 communicate via connection 829 with other processors and devices through LAN 840 and through the Internet 880.

Software components 802 include an operating system 850 and a set of tasks under control of operating system 310, such as a Java™ application program 860 and, importantly, JVM software 354 and kernel 355. Operating system 310 also allows processor 805 to control various devices such as persistent storage 808, user I/O 820, and network interface 825. Processor 805 executes the software of operating system 310, application 860, JVM 354 and kernel 355 in conjunction with memory 806 and other components of computer system 800. In an embodiment, software 802 includes network software 302a, JVM1, RJVM2 and RJVM3, as illustrated in server 302 of FIG. 3c. In an embodiment, Java™ application program 860 is Java™ application 302c as illustrated in FIG. 3c.

Persons of ordinary skill in the art will appreciate that the system of FIG. 8 is intended to be illustrative, not restrictive, and that a wide variety of computational, communications, and information devices can be used in place of or in addition to what is shown in FIG. 8. For example, connections through the Internet 880 generally involve packet switching by intermediate router computers (not shown), and computer 800 is likely to access any number of Web servers, including but by no means limited to computer 800 and Web server 881, during a typical Web client session.

The foregoing description of the preferred embodiments of the present invention has been provided for the purposes of illustration and description. It is not intended to be exhaustive or to limit the invention to the precise forms disclosed. Obviously, many modifications and variations will be apparent to practitioners skilled in the art. The embodiments were chosen and described in order to best explain the principles of the invention and its practical applications, thereby enabling others skilled in the art to understand the invention for various embodiments and with the various modifications as are suited to the particular use contemplated. It is intended that the scope of the invention be defined by the following claims and their equivalents.

What is claimed is:

1. An article of manufacture including an information storage medium wherein is stored information, comprising:
 - a first set of digital information, including a Java™ virtual machine with a stub having a load balancing software component for selecting a service provider from a plurality of service providers and a failover software component for removing a failed service provider from a list identifying the plurality of service providers, wherein the Java™ virtual machine with the stub is located on a client processing device,
 - wherein the load balancing software selects a particular service provider, from the list of plurality of service providers, if both an affinity exists for the particular service provider and the particular service provider provides a service requested, and,
 - wherein an affinity exists for a particular service provider when that particular service provider, or the server associated with the service provider, is currently participating in a transaction between either of the service provider or server and the client processing device.

19

2. The article of manufacture of claim 1, wherein if no affinity for a service provider exists, or if the particular service provider for which the affinity exists does not provide the service requested, then the load balancing software component selects a service provider from the list of plurality of service providers in a round robin manner.

3. The article of manufacture of claim 1, wherein if no affinity for a service provider exists, or if the particular service provider for which the affinity exists does not provide the service requested, then the load balancing software component randomly selects a service provider from the list of service providers.

4. The article of manufacture of claim 1, wherein if no affinity for a service provider exists, or if the particular service provider for which the affinity exists does not provide the service requested, then the load balancing software component selects a service provider from the list of service providers based upon the load of each service provider.

5. The article of manufacture of claim 1, wherein if no affinity for a service provider exists, or if the particular service provider for which the affinity exists does not provide the service requested, then the load balancing software component selects a service provider from the list of service providers based upon the data type requested.

6. The article of manufacture of claim 1, wherein if no affinity for a service provider exists, or if the particular service provider for which the affinity exists does not provide the service requested, then the load balancing software component selects a service provider from the list of service providers based upon the closest physical service provider.

7. The article of manufacture of claim 1, wherein if no affinity for a service provider exists, or if the particular service provider for which the affinity exists does not provide the service requested, then the load balancing software component selects a service provider from the list of service providers based upon a time period in which each service provider responds.

8. An article of manufacture including an information storage medium wherein is stored information, comprising:

a first set of digital information, including a Java™ virtual machine with a Java™ bean object for selecting a service provider from a plurality of service providers; wherein the Java™ bean object has a load balancing software component that selects a particular service provider, from the plurality of service providers, if both an affinity exists for the particular service provider and the particular service provider provides a service requested, and,

wherein an affinity exists for a particular service provider when that particular service provider, or the server associated with the service provider, is currently participating in a transaction between either of the service provider or server and the client processing device.

9. The article of manufacture of claim 8, wherein the Java™ bean object has a failover software component for removing a failed service provider from a list of service providers.

10. The article of manufacture of claim 8, wherein if no affinity for a service provider exists, or if the particular service provider for which the affinity exists does not provide the service requested, then the load balancing software component selects a service provider from the plurality of service providers in a round robin manner.

11. The article of manufacture of claim 8, wherein if no affinity for a service provider exists, or if the particular

20

service provider for which the affinity exists does not provide the service requested, then the load balancing software component randomly selects a service provider.

12. The article of manufacture of claim 8, wherein if no affinity for a service provider exists, or if the particular service provider for which the affinity exists does not provide the service requested, then the load balancing software component selects a service provider from the plurality of service providers based upon the load of each service provider.

13. The article of manufacture of claim 8, wherein if no affinity for a service provider exists, or if the particular service provider for which the affinity exists does not provide the service requested, then the load balancing software component selects a service provider from the plurality of service providers based upon the data type requested.

14. The article of manufacture of claim 8, wherein if no affinity for a service provider exists, or if the particular service provider for which the affinity exists does not provide the service requested, then the load balancing software component selects a service provider from the plurality of service providers based upon the closest physical service provider.

15. The article of manufacture of claim 8, wherein if no affinity for a service provider exists, or if the particular service provider for which the affinity exists does not provide the service requested, then the load balancing software component selects a service provider from the plurality of service providers based upon a time period in which each service provider responds.

16. The article of manufacture of claim 8, further comprising:

a second set of digital information, including a stateless session bean.

17. The article of manufacture of claim 8, further comprising:

a second set of digital information, including a stateful session bean.

18. The article of manufacture of claim 8, further comprising:

a second set of digital information, including an entity session bean.

19. An apparatus, comprising:

a processor;

an instruction store, coupled to the processor, comprising an article of manufacture as recited in claim 1; and a data store, coupled to the processor, wherein an application program can be stored.

20. An apparatus, comprising:

a processor;

an instruction store, coupled to the processor, comprising an article of manufacture as recited in claim 8; and a data store, coupled to the processor, wherein an application program can be stored.

21. A processing device implemented method, comprising the steps of:

obtaining, by a stub, a list of service providers; and selecting a service provider for use in a transaction, by the stub, from the list of service providers

wherein the selecting step includes selecting, by the stub, a particular service provider, from the list of plurality of service providers, if both an affinity exists for the particular service provider and the particular service provider provides a service requested, and,

wherein an affinity exists for a particular service provider when that particular service provider, or the server

21

associated with the service provider, is currently participating in the transaction.

22. The method of claim 21, wherein the list of service providers is obtained from a naming service.

23. The method of claim 21, wherein the list of service providers is obtained from a naming service.

24. The method of claim 21, wherein if no affinity for a service provider exists, or if the particular service provider for which the affinity exists does not provide the service requested, then the step of selecting further includes the step of:

selecting a service provider, by the stub, from the list of service providers in a round robin manner.

25. The method of claim 21, wherein if no affinity for a service provider exists, or if the particular service provider for which the affinity exists does not provide the service requested, then the step of selecting further includes the step of:

selecting a service provider, by the stub, randomly from the list of service providers.

26. The method of claim 21, wherein if no affinity for a service provider exists, or if the particular service provider for which the affinity exists does not provide the service requested, then the step of selecting further includes the steps of:

obtaining the load of each service provider, by the stub, in the list of service providers; and,

selecting a service provider, by the stub, based upon the load of each service provider.

27. The method of claim 21, wherein if no affinity for a service provider exists, or if the particular service provider for which the affinity exists does not provide the service requested, then the step of selecting further includes the steps of:

determining the type of data requested; and,

selecting a service provider, by the stub, from the list of service providers based upon the data type.

28. The method of claim 21, wherein if no affinity for a service provider exists, or if the particular service provider for which the affinity exists does not provide the service requested, then the step of selecting further includes the steps of:

determining the physical distance to each service provider, by the stub, in the list of service providers; and,

selecting a service provider, by the stub, from the list of service providers based upon on the closest physical distance to the service provider.

29. The method of claim 21, wherein if no affinity for a service provider exists, or if the particular service provider for which the affinity exists does not provide the service requested, then the step of selecting further includes the steps of:

determining a time period for each service provider, by the stub, in the list of service providers to respond; and,

selecting a service provider from the list of service providers based upon the time period for each service provider to respond.

30. A processing device implemented method, comprising:

obtaining a calling thread;

determining, by a client, if the calling thread has an affinity for a server, wherein an affinity exists for a particular server when that particular server is currently participating in a transaction between the server and the client;

22

determining if the server provides a service;

obtaining a list of services, wherein the service is in the list of services providers; and,

attempting to obtain the service.

31. The method of claim 30, further comprising:

obtaining a failover method if the service is not available.

32. The method of claim 31, wherein the failover method obtains a next service provider in the list of service providers.

33. The method of claim 31, wherein the failover method obtains a randomly selected service provider in the list of service providers.

34. The method of claim 31, wherein the failover method obtains a service provider with the least load in the list of service providers.

35. The method of claim 31, wherein the failover method obtains a service provider based on a data type in the list of service providers.

36. The method of claim 31, wherein the failover method obtains a service provider based on the closest physical distance to the service provider.

37. The method of claim 31, wherein the failover method obtains a service provider based on a time for a response from a service provider in the plurality of service providers.

38. A method of providing failover in a distributed processing system, to select which service provider within a plurality of service providers should respond to a request from a client to access a service, the method comprising the steps of:

determining whether the client has an affinity for a particular service provider within the plurality of service providers;

if the client does have an affinity for a particular service provider then the substeps of

determining whether the particular service provider can provide the service requested; and,

returning the name of that service provider to the client;

if the client does not have an affinity for a particular service provider, or if the particular service provider is no longer available to provide the service requested, then the substeps of

selecting a new service provider from the plurality of service providers; and,

returning the name of the new service provider to the client; and,

allowing the client to access the service using the named service provider, if the named service provider exists and can provide the service.

39. The method of claim 38 wherein said step (B) of determining whether the client has an affinity for a particular service provider includes determining which service provider is coordinating the current transaction between the client and the server, and identifying that service provider as the particular service provider which the client has an affinity for.

40. The method of claim 38 wherein said step of selecting a new service provider includes selecting a new service provider from a list of service providers.

41. The method of claim 38 wherein said step of selecting a new service provider includes selecting a service provider from a list of service providers which is maintained by a naming service.

42. The method of claim 38 wherein said step of selecting a new service provider includes calling a `get.next.provider` function to obtain and select the next service provider on the list of service providers.

23

43. The method of claim 38 further comprising, following said step (D) of allowing the client to request service from the named service provider, the additional steps of:

(E) if the named service provider does not exist or cannot provide the service, then calling a failover method that includes the substeps of
 identifying the named service provider as a failed service provider,
 selecting a failover service provider from the plurality of service providers, and,
 returning the name of the failover service provider to the client.

44. The method of claim 38 further comprising the step of: removing the failed service provider from a list of available service providers within the distributed processing system.

45. The method of claim 38 wherein the service is a database or file system.

46. The method of claim 38, wherein if no affinity for a service provider exists, or if the particular service provider for which the affinity exists does not provide the service requested, then the system selects a service provider from the list of plurality of service providers in a round robin manner.

47. The method of claim 38, wherein if no affinity for a service provider exists, or if the particular service provider for which the affinity exists does not provide the service requested, then the system randomly selects a service provider from the list of service providers.

48. The method of claim 38, wherein if no affinity for a service provider exists, or if the particular service provider for which the affinity exists does not provide the service requested, then the system selects a service provider from the list of service providers based upon the load of each service provider.

49. The method of claim 38, wherein if no affinity for a service provider exists, or if the particular service provider for which the affinity exists does not provide the service requested, then the system selects a service provider from the list of service providers based upon the data type requested.

50. The method of claim 38, wherein if no affinity for a service provider exists, or if the particular service provider for which the affinity exists does not provide the service requested, then the system selects a service provider from the list of service providers based upon the closest physical service provider.

51. The method of claim 38, wherein if no affinity for a service provider exists, or if the particular service provider for which the affinity exists does not provide the service requested, then the system selects a service provider from the list of service providers based upon a time period in which each service provider responds.

52. A method of using load balancing and/or failover in a distributed processing system to select which a service provider within a plurality of service providers can respond to a transaction request from a client to access a service, the method comprising the steps of:

(A) receiving a transaction request from a client to access a service;

(B) determining whether the client has an affinity for a particular service provider within said plurality of service providers;

(C1) if the client does have an affinity for a particular service provider then the substeps of
 determining whether the particular service provider can provide the service requested, and,

24

returning the name of the particular service provider to the client for use by the client in accessing the service;

(C2) if the client does not have an affinity for a particular service provider, then the substeps of
 selecting a new service provider from the plurality of service providers,
 determining whether the new service provider can provide the service requested, and,
 returning the name of the new service provider to the client for use by the client in accessing the service; and,

(D) allowing the client to request service from the named service provider, if the named service provider is available and can provide access to the service requested.

53. The method of claim 52 wherein said step (B) of determining whether the client has an affinity for a particular service provider includes determining which service provider is coordinating the current transaction between the client and the server, and identifying that service provider as the particular service provider which the client has an affinity for.

54. The method of claim 52 wherein said step of selecting a new service provider includes selecting a new service provider from a list of service providers.

55. The method of claim 52 wherein said step of selecting a new service provider includes selecting a service provider from a list of service providers which is maintained by a naming service.

56. The method of claim 52 wherein said step of selecting a new service provider includes calling a `get.next.provider` function to obtain and select the next service provider on the list of service providers.

57. The method of claim 52 further comprising, following said step (D) of allowing the client to request service from the named service provider, the additional steps of:

(E) if the named service provider does not exist or cannot provide the service, then calling a failover method that includes the substeps of
 identifying the named service provider as a failed service provider,
 selecting a failover service provider from the plurality of service providers, and,
 returning the name of the failover service provider to the client.

58. The method of claim 52 further comprising the step of: removing the failed service provider from a list of available service providers within the distributed processing system.

59. The method of claim 52 wherein the service is a database or file system.

60. The method of claim 52, wherein if no affinity for a service provider exists, or if the particular service provider for which the affinity exists does not provide the service requested, then the system selects a service provider from the list of plurality of service providers in a round robin manner.

61. The method of claim 52, wherein if no affinity for a service provider exists, or if the particular service provider for which the affinity exists does not provide the service requested, then the system randomly selects a service provider from the list of service providers.

62. The method of claim 52, wherein if no affinity for a service provider exists, or if the particular service provider for which the affinity exists does not provide the service requested, then the system selects a service provider from

25

the list of service providers based upon the load of each service provider.

63. The method of claim 52, wherein if no affinity for a service provider exists, or if the particular service provider for which the affinity exists does not provide the service requested, then the system selects a service provider from the list of service providers based upon the data type requested.

64. The method of claim 52, wherein if no affinity for a service provider exists, or if the particular service provider for which the affinity exists does not provide the service requested, then the system selects a service provider from the list of service providers based upon the closest physical service provider.

65. The method of claim 52, wherein if no affinity for a service provider exists, or if the particular service provider for which the affinity exists does not provide the service requested, then the system selects a service provider from the list of service providers based upon a time period in which each service provider responds.

66. A system for load balancing and failover of requests by a client to access a service in a distributed processing system comprising:

a handler for receiving requests from a client to access a service;

a plurality of service providers for providing client access to the service;

software code that performs the method of

determining whether the client has an affinity for a particular service provider within the plurality of service providers;

if the client does have an affinity for a particular service provider then the substeps of

determining whether the particular service provider can provide the service requested, and, returning the name of that service provider to the client;

if the client does not have an affinity for a service provider, then the substeps of selecting a new service provider from the plurality of service providers, and,

returning the name of the new service provider to the client; and,

allowing the client to access the service using the named service provider, if the named service provider exists and can provide the service.

26

67. The system of claim 66 further comprising:

a list of currently available service providers, and, wherein a failed service provider is removed from the list of service providers.

68. The system of claim 66 wherein the service is a database or file system.

69. The system of claim 66, wherein if no affinity for a service provider exists, or if the particular service provider for which the affinity exists does not provide the service requested, then the system selects a service provider from the list of plurality of service providers in a round robin manner.

70. The system of claim 66, wherein if no affinity for a service provider exists, or if the particular service provider for which the affinity exists does not provide the service requested, then the system randomly selects a service provider from the list of service providers.

71. The system of claim 66, wherein if no affinity for a service provider exists, or if the particular service provider for which the affinity exists does not provide the service requested, then the system selects a service provider from the list of service providers based upon the load of each service provider.

72. The system of claim 66, wherein if no affinity for a service provider exists, or if the particular service provider for which the affinity exists does not provide the service requested, then the system selects a service provider from the list of service providers based upon the data type requested.

73. The system of claim 66, wherein if no affinity for a service provider exists, or if the particular service provider for which the affinity exists does not provide the service requested, then the system selects a service provider from the list of service providers based upon the closest physical service provider.

74. The system of claim 66, wherein if no affinity for a service provider exists, or if the particular service provider for which the affinity exists does not provide the service requested, then the system selects a service provider from the list of service providers based upon a time period in which each service provider responds.

* * * * *

UNITED STATES PATENT AND TRADEMARK OFFICE
CERTIFICATE OF CORRECTION

PATENT NO. : 6,581,088 B1
DATED : June 17, 2003
INVENTOR(S) : Dean B. Jacobs and Eric M. Halpern

Page 1 of 1

It is certified that error appears in the above-identified patent and that said Letters Patent is hereby corrected as shown below:

Column 21.

Lines 3 and 4, please replace Claim 22 with the following:

22. The method of claim 21, further comprising the step of: removing a failed service provider, by the stub, from the list of service providers.

Signed and Sealed this

Seventh Day of October, 2003

A handwritten signature in black ink, appearing to read "James E. Rogan", with a horizontal line drawn underneath it.

JAMES E. ROGAN
Director of the United States Patent and Trademark Office

UNITED STATES PATENT AND TRADEMARK OFFICE
CERTIFICATE OF CORRECTION

PATENT NO. : 6,581,088 B1
DATED : June 17, 2003
INVENTOR(S) : Dean Bernard Jacobs and Eric M. Halpern

Page 1 of 1

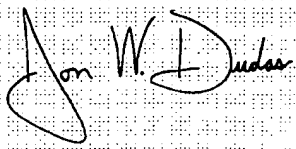
It is certified that error appears in the above-identified patent and that said Letters Patent is hereby corrected as shown below:

Title page.

Item [73], Assignee: "Beas Systems, Inc." should be -- BEA SYSTEMS, INC. --.

Signed and Sealed this

Tenth Day of August, 2004

A handwritten signature in black ink, reading "Jon W. Dudas", is written over a rectangular area of fine, dotted grid lines.

JON W. DUDAS
Acting Director of the United States Patent and Trademark Office



US00599979A

United States Patent [19]

Vellanki et al.

[11] **Patent Number:** 5,999,979[45] **Date of Patent:** Dec. 7, 1999

[54] **METHOD AND APPARATUS FOR DETERMINING A MOST ADVANTAGEOUS PROTOCOL FOR USE IN A COMPUTER NETWORK**

[75] Inventors: **Srinivas Prasad Vellanki**, Milpitas; **Anthony William Cannon**, Mountain View; **Hemanth Srinivas Ravi**, Milpitas; **Anders Edgar Klemets**, Sunnyvale, all of Calif.

[73] Assignee: **Microsoft Corporation**, Redmond, Wash.

[21] Appl. No.: **08/818,769**

[22] Filed: **Mar. 14, 1997**

Related U.S. Application Data

[60] Provisional application No. 60/036,661, Jan. 30, 1997, and provisional application No. 60/036,662, Jan. 30, 1997.

[51] Int. Cl.⁶ **G06F 15/16**

[52] U.S. Cl. **709/232; 709/237**

[58] Field of Search 395/200.57, 200.58, 395/200.6, 200.76, 200.62; 710/11; 709/228, 227, 230, 231, 232, 237

[56] **References Cited**

U.S. PATENT DOCUMENTS

4,931,250 6/1990 Greszczuk 375/8
5,202,899 4/1993 Walsh 375/8

5,349,649 9/1994 Iijima 395/275
5,557,724 9/1996 Sampat et al. 395/157
5,687,174 11/1997 Edem et al. 370/446

Primary Examiner—Stuart S. Levy

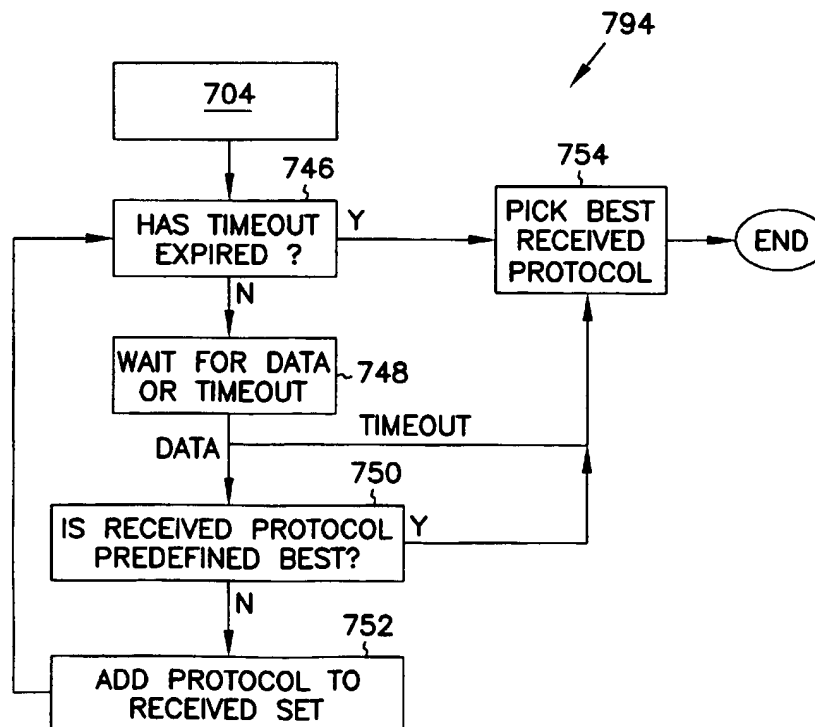
Assistant Examiner—Kenneth W. Fields

Attorney, Agent, or Firm—Schwegman, Lundberg, Woessner & Kluth P.A.

[57] **ABSTRACT**

A method in a computer network for automatically detecting a most advantageous protocol for communication by a client computer, said client computer being configured to be coupled to a server computer via a computer network. The method includes initiating a plurality of protocol threads for sending from the client computer to the server computer, a plurality of data requests. Each of the data requests employs a different protocol and a different connection. The data requests are configured to solicit, responsive to the data requests, a set of responses from the server computer. Each of the responses employs a protocol associated with a respective one of the data requests. The method further includes receiving at the client computer at least a subset of the responses. The method also includes initiating a control thread at the client computer. The control thread monitors the subset of the responses as each response is received from the server computer to select the most advantageous protocol from protocols associated with the subset of the responses, wherein the most advantageous protocol is determined based on a predefined protocol priority.

23 Claims, 11 Drawing Sheets



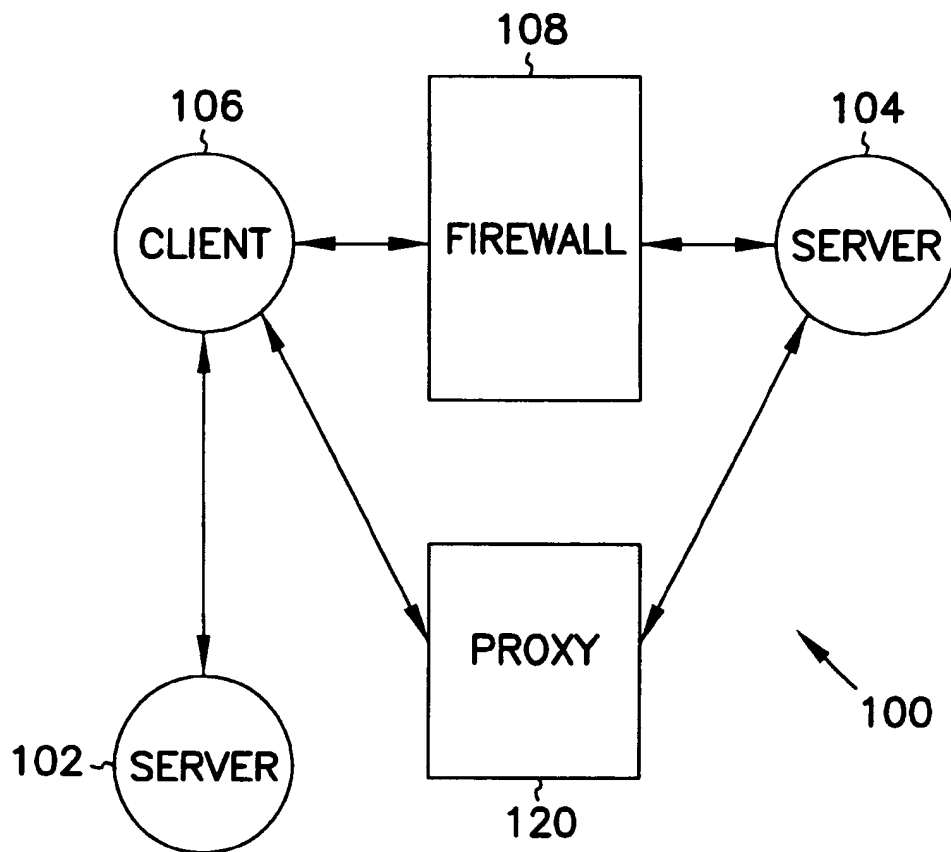


FIG. 1

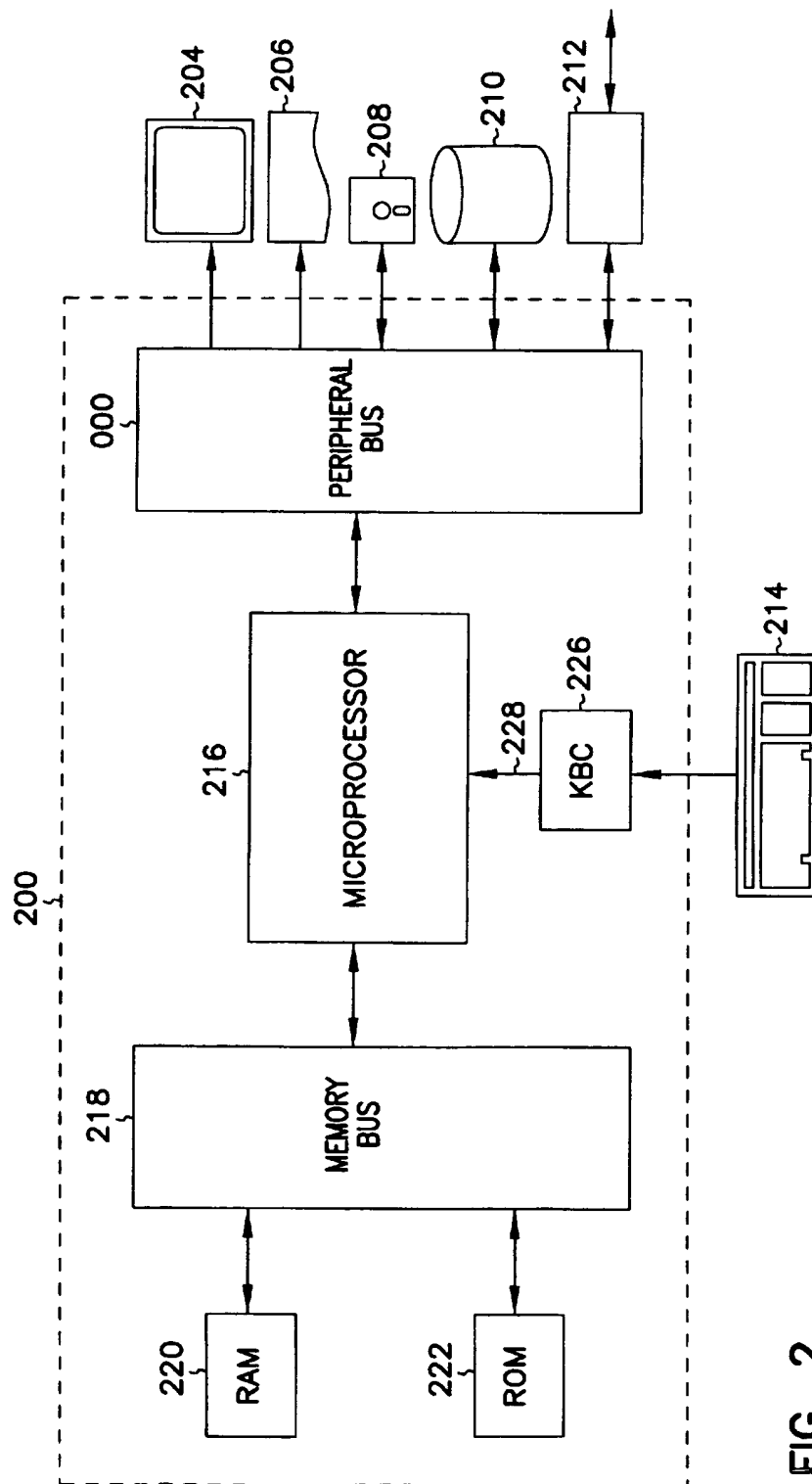
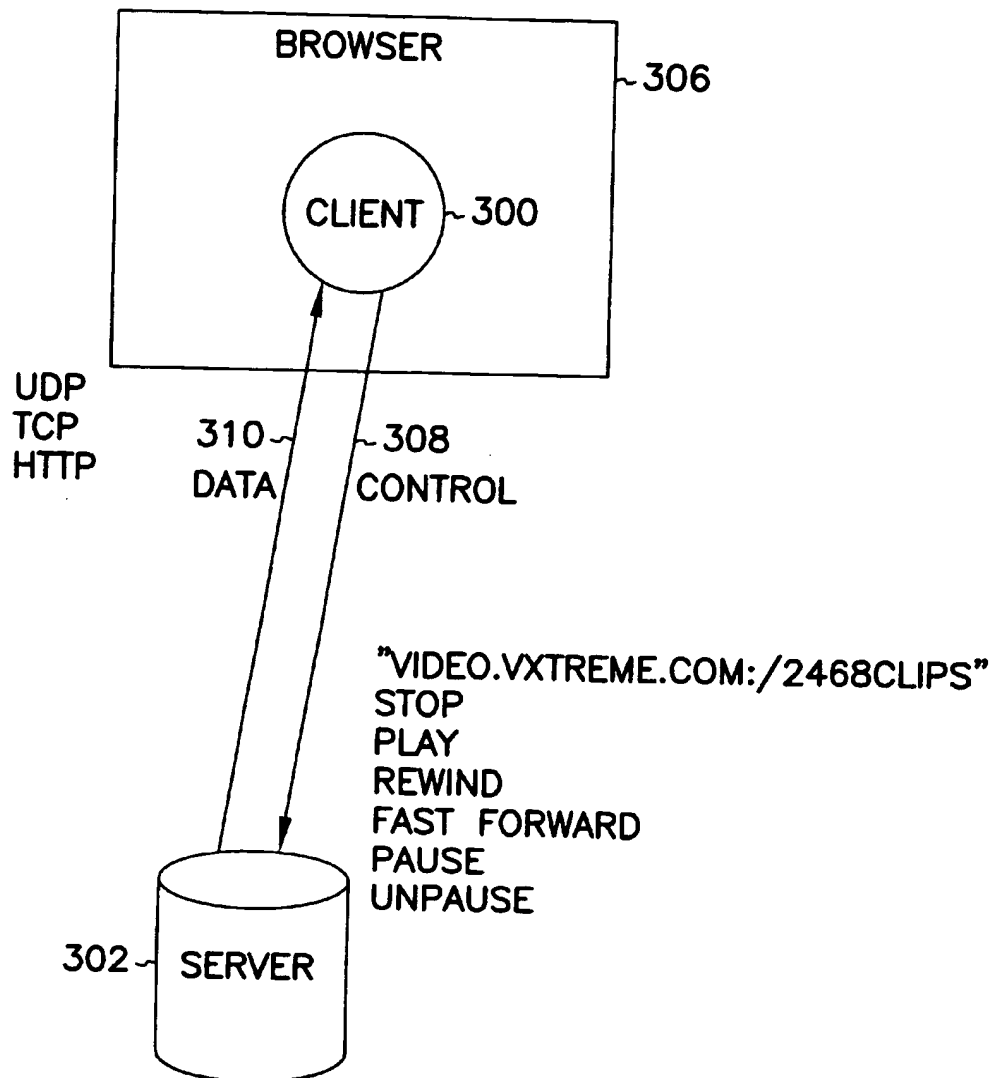
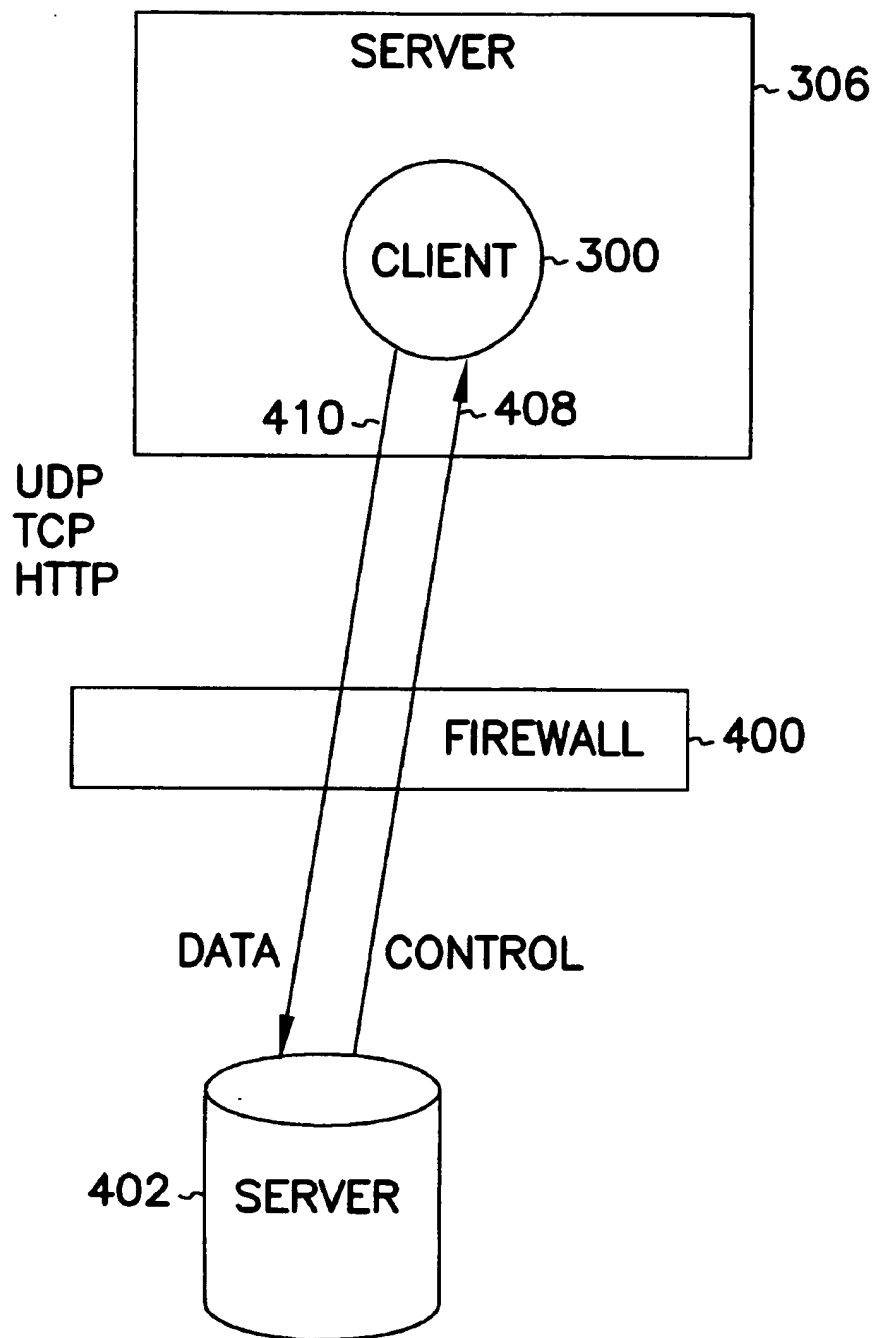


FIG. 2

**FIG. 3**

**FIG. 4**

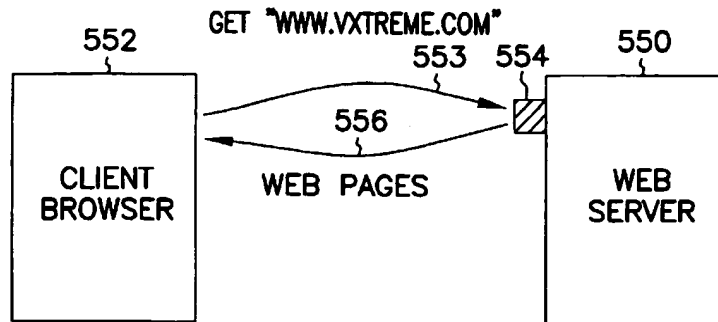


FIG. 5A (PRIOR ART)

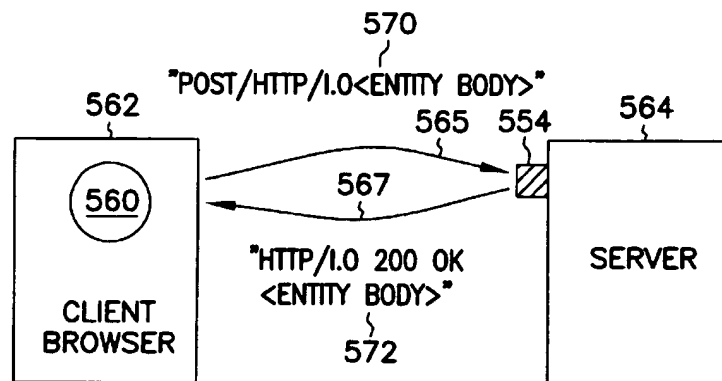


FIG. 5B

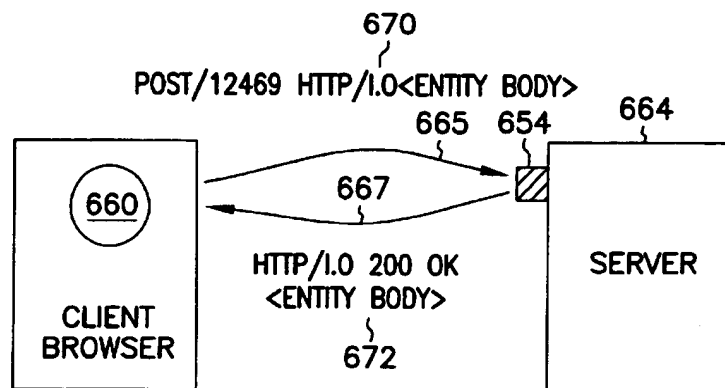


FIG. 5D

<ENTITY BODY>:
AUDIO:29,999
OR
VIDEO : 35,122

672

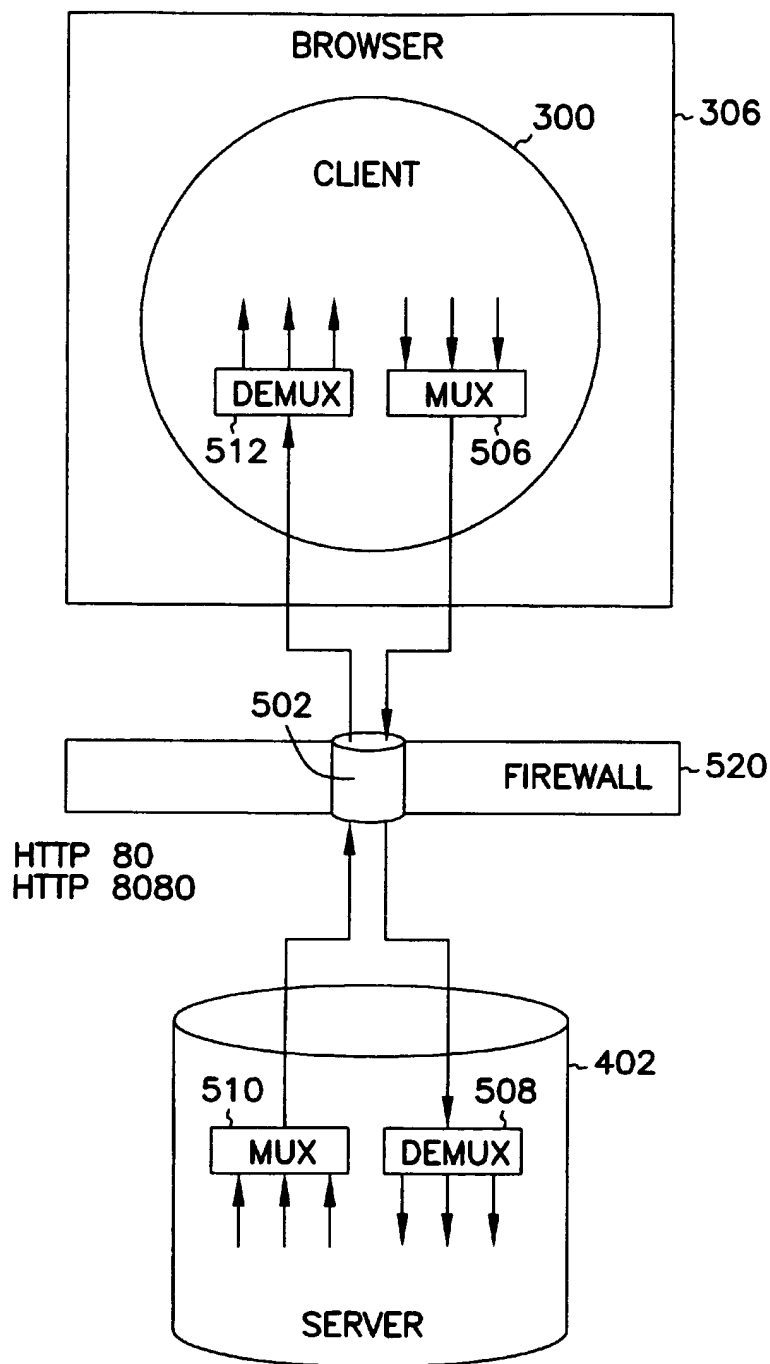
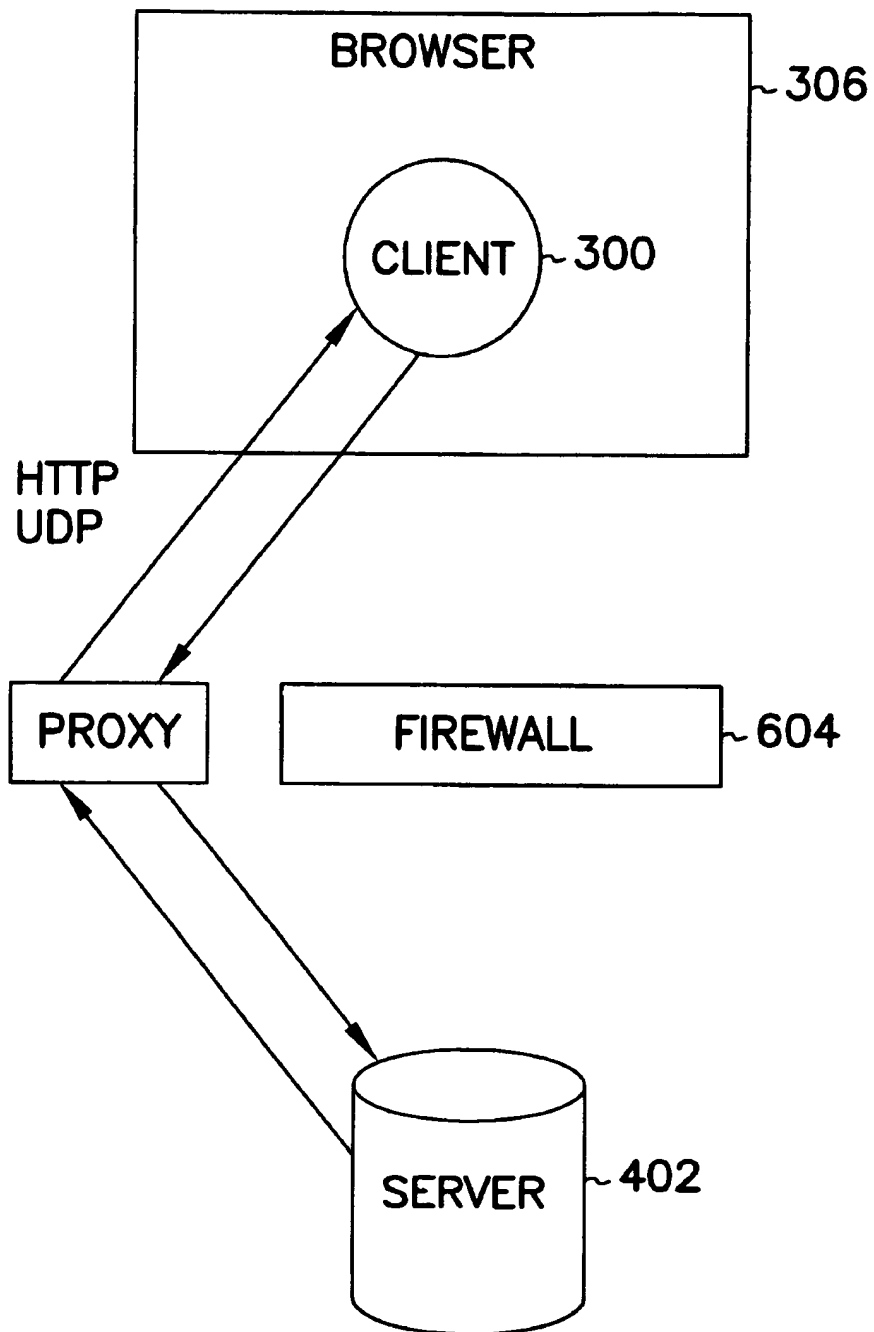


FIG. 5C

**FIG. 6**

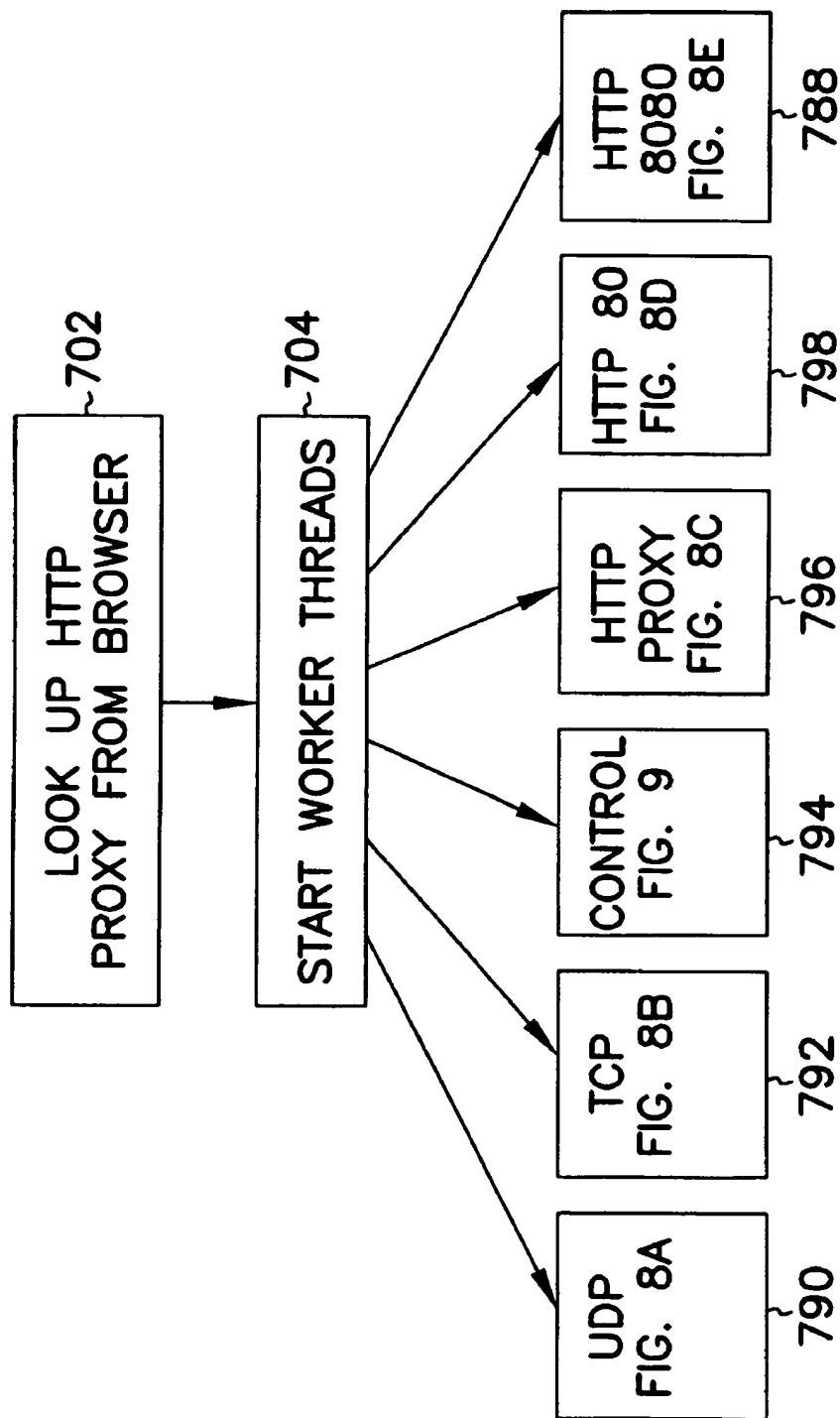


FIG. 7

FIG. 8A

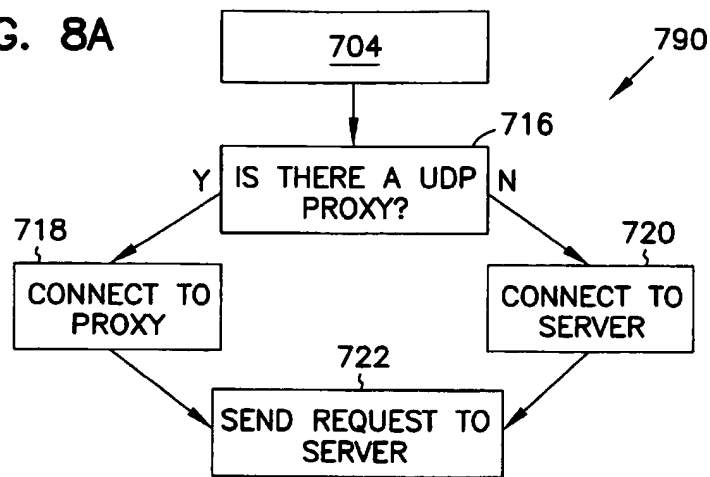


FIG. 8B

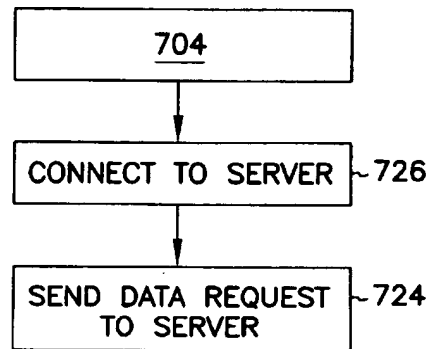


FIG. 8C

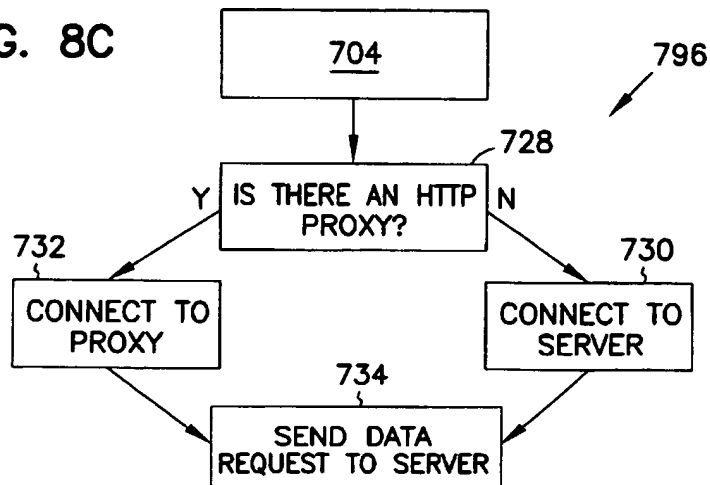


FIG. 8D

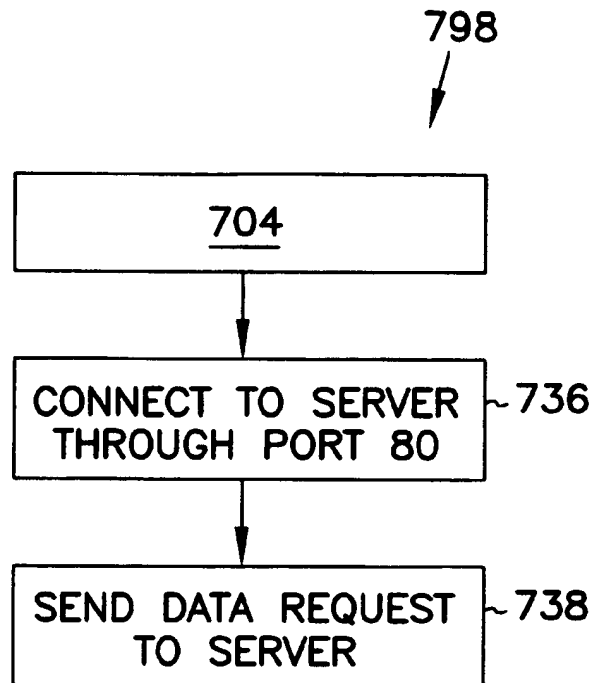
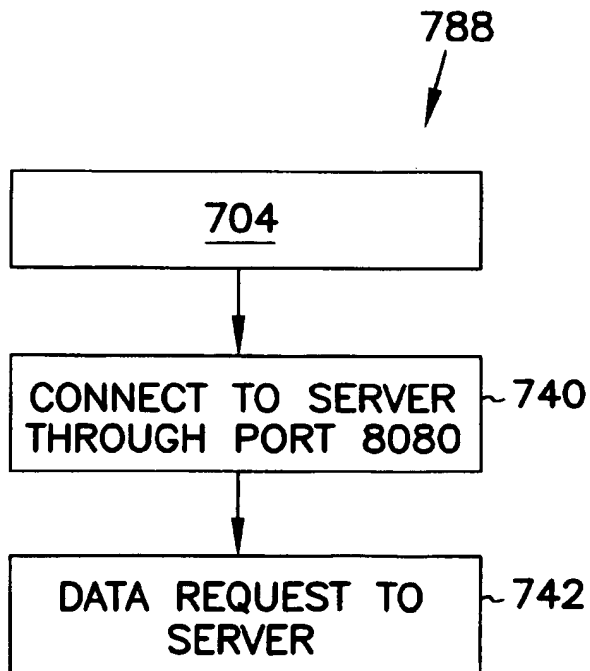


FIG. 8E



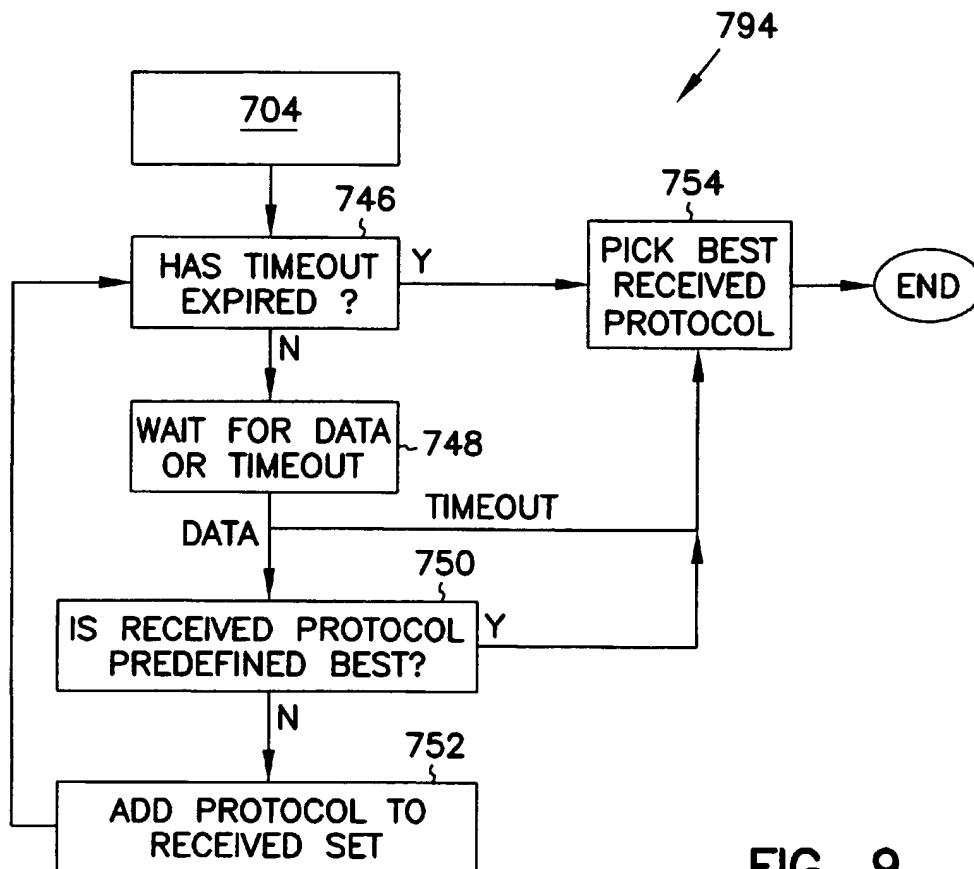


FIG. 9

METHOD AND APPARATUS FOR DETERMINING A MOST ADVANTAGEOUS PROTOCOL FOR USE IN A COMPUTER NETWORK

This application claims priority under 35 U.S.C 119 (e) of a provisional application entitled "VCR CONTROL FUNCTIONS" filed Jan. 30, 1997 by inventors Anthony W. Cannon, Anders E. Klemets, Hemanth S. Ravi, and David del Val (application Ser. No. 60/036,661) and a provisional application entitled "METHODS AND APPARATUS FOR AUTODETECTING PROTOCOLS IN A COMPUTER NETWORK" filed Jan. 30, 1997 by inventors Anthony W. Cannon, Anders E. Klemets, Hemanth S. Ravi, and David del Val (application Ser. No. 60/036,662).

CROSS REFERENCE TO RELATED APPLICATIONS

This application is related to co-pending U.S. application Ser. No. 08/818,805, filed on Mar. 14, 1997, entitled "Method and Apparatus for Implementing Motion Detection in Video Compression", U.S. application Ser. No. 08/819,507, filed Mar. 14, 1997, entitled "Digital Video Signal Encoder and Encoding Method", U.S. application Ser. No. 08/818,804, filed on Mar. 14, 1997, entitled "Production of a Video Stream with Synchronized Annotations over a Computer Network", U.S. application Ser. No. 08/819,586, filed on Mar. 14, 1997, entitled "Method and Apparatus for Implementing Control Functions in a Streamed Video Display System", U.S. application Ser. No. 08/818,769, filed on Mar. 14, 1997, entitled "Method and Apparatus for Automatically Detecting Protocols in a Computer Network," U.S. application Ser. No. 08/818,127, filed on Mar. 14, 1997, entitled "Dynamic Bandwidth Selection for Efficient Transmission of Multimedia Streams in a Computer Network," U.S. application Ser. No. 08/819,585, filed on Mar. 14, 1997, entitled "Streaming and Display of a Video Stream with Synchronized Annotations over a Computer Network", U.S. application Ser. No. 08/818,664, filed on Mar. 14, 1997, entitled "Selective Retransmission for Efficient and Reliable Streaming of Multimedia Packets in a Computer Network", U.S. application Ser. No. 08/819,579, filed Mar. 14, 1997, entitled "Method and Apparatus for Table-Based Compression with Embedded Coding", U.S. application Ser. No. 08/818,826, filed on Mar. 14, 1997, entitled "Digital Video Signal Encoder and Encoding Method", all filed concurrently herewith, U.S. application Ser. No. 08/822,156, filed on Mar. 17, 1997, entitled "Method and Apparatus for Communication Media Commands and Data Using the HTTP Protocol", provisional U.S. application Ser. No. 60/036,662, filed on Jan. 30, 1997, entitled "Methods and Apparatus for Autodetecting Protocols in a Computer Network" U.S. application Ser. No. 08/625,650, filed on Mar. 29, 1996, entitled "Table-Based Low-Level Image Classification System", U.S. application Ser. No. 08/714,447, filed on Sep. 16, 1996, entitled "Multimedia Compression System with Additive Temporal Layers", and is a continuation-in-part of U.S. application Ser. No. 08/623,299, filed on Mar. 28, 1996, entitled "Table-Based Compression with Embedded Coding", which are all incorporated by reference in their entirety for all purposes.

BACKGROUND OF THE INVENTION

The present invention relates to data communication in a computer network. More particularly, the present invention relates to improved methods and apparatus for permitting a

client computer in a client-server architecture computer network to automatically detect the most advantageous protocol, among the protocols available, for use in communicating with the server irrespective whether there exist firewalls or proxies in the network.

Client-server architectures are well known to those skilled in the computer art. For example, in a typical computer network, one or more client computers may be coupled to any number of server computers. Client computers typically refer to terminals or personal computers through which end users interact with the network. Server computers typically represent nodes in the computer network where data, application programs, and the like, reside. Server computers may also represent nodes in the network for forwarding data, programs, and the likes from other servers to the requesting client computers.

To facilitate discussion, FIG. 1 illustrates a computer network 100, representing for example a subset of an international computer network popularly known as the Internet. As is well known, the Internet represents a well-known international computer network that links, among others, various military, governmental, educational, nonprofit, industrial and financial institutions, commercial enterprises, and individuals. There are shown in FIG. 1 a server 102, a server 104, and a client computer 106. Server computer 104 is separated from client computer 106 by a firewall 108, which may be implemented in either software or hardware, and may reside on a computer and/or circuit between client computer 106 and server computer 104.

Firewall 108 may be specified, as is well known to those skilled in the art, to prevent certain types of data and/or protocols from traversing through it. The specific data and/or protocols prohibited or permitted to traverse firewall 108 depend on the firewall parameters, which are typically set by a system administrator responsible for the maintenance and security of client computer 106 and/or other computers connected to it, e.g., other computers in a local area network. By way of example, firewall 108 may be set up to prevent TCP, UDP, or HTTP (Transmission Control Protocol, User Datagram Protocol, and Hypertext Transfer Protocol, respectively) data and/or other protocols from being transmitted between client computer 106 and server 104. The firewalls could be configured to allow specific TCP or UDP sessions, for example outgoing TCP connection to certain ports, UDP sessions to certain ports, and the like.

Without a firewall, any type of data and/or protocol may be communicated between a client computer and a server computer if appropriate software and/or hardware are employed. For example, server 102 resides on the same side of firewall 108 as client computer 106, i.e., firewall 108 is not disposed in between the communication path between server 102 and client computer 106. Accordingly, few, if any, of the protocols that client computer 106 may employ to communicate with server 102 may be blocked.

As is well known to those skilled in the art, some computer networks may be provided with proxies, i.e., software codes or hardware circuitries that facilitate the indirect communication between a client computer and a server around a firewall. With reference to FIG. 1, for example, client computer 106 may communicate with server 104 through proxy 120. Through proxy 120, HTTP data, which may otherwise be blocked by firewall 108 for the purpose of this example, may be transmitted between client computer 106 and server computer 104.

In some computer networks, one or more protocols may be available for communication between the client computer

3

and the server computer. For certain applications, one of these protocols, however, is often more advantageous, i.e., suitable, than others. By way of example, in applications involving real-time data rendering (such as rendering audio, video, and/or annotation data as they are streamed from a server, as described in above-referenced U.S. patent application Ser. Nos. 08/818,804 and 08/819,585 (Atty: Docket No.: VXT710)), it is highly preferable that the client computer executing that application selects a protocol that permits the greatest degree of control over the transmission of data packets and/or enables data transmission to occur at the highest possible rate. This is because these applications are fairly demanding in terms of their bit rate and connection reliability requirements. Accordingly, the quality of the data rendered, e.g., the video and/or audio clips played, often depends on whether the user has successfully configured the client computer to receive data from the server computer using the most advantageous protocol available.

In the prior art, the selection of the most advantageous protocol for communication between client computer 106 and server computer 104 typically requires a high degree of technical sophistication on the part of the user of client computer 106. By way of example, it is typically necessary in the prior art for the user of client computer 106 to understand the topology of computer network 100, the protocols available for use with the network, and/or the protocols that can traverse firewall 108 before that user can be expected to configure his client computer 106 for communication.

This level of technical sophistication is, however, likely to be beyond that typically possessed by an average user of client computer 106. Accordingly, users in the prior art often find it difficult to configure their client computers even for simple communication tasks with the network. The difficulties may be encountered for example during the initial setup or whenever there are changes in the topology of computer network 100 and/or in the technology employed to transmit data between client computer 106 and server 104. Typically, expert and expensive assistance is required, if such assistance is available at all in the geographic area of the user.

Furthermore, even if the user can configure client computer 106 to communicate with server 104 through firewall 108 and/or proxy 120, there is no assurance that the user of client computer 106 has properly selected, among the protocols available, the most advantageous protocol communication (e.g., in terms of data transmission rate, transmission control, and the like). As mentioned earlier, the ability to employ the most advantageous protocol for communication, while desirable for most networking applications, and is particularly critical in applications such as real-time data rendering (e.g., rendering of audio, video, and/or annotation data as they are received from the server). If a less than optimal protocol is chosen for communication, the quality of the rendered data, e.g., the video clips and/or audio clips, may suffer.

In view of the foregoing, there are desired improved techniques for permitting a client computer in a client-server network to efficiently, automatically, and appropriately select the most advantageous protocol to communicate with a server computer.

SUMMARY OF THE INVENTION

The invention relates, in one embodiment, to a method in a computer network for automatically detecting a most advantageous protocol for communication by a client computer. The client computer is configured to be coupled to a

4

server computer via a computer network. The method includes sending from the client computer to the server computer, a plurality of data requests. Each of the data requests employs a different protocol and a different connection. The data requests are configured to solicit, responsive to the data requests, a set of responses from the server computer. Each of the responses employs a protocol associated with a respective one of the data requests.

The method further includes receiving at the client computer at least a subset of the responses. The method also includes monitoring the subset of the responses as each response is received from the server computer to select the most advantageous protocol from protocols associated with the subset of the responses, wherein the most advantageous protocol is determined based on a predefined protocol priority.

In another embodiment, the invention relates to a method in a computer network for automatically detecting a most advantageous protocol for communication by a client computer. The client computer is configured to be coupled to a server computer via a computer network. The method includes sending from the client computer to the server computer, a plurality of data requests. Each of the data requests employs a different protocol and a different connection.

The method further includes receiving at least a subset of the data requests at the server computer. The method additionally includes sending a set of responses from the server computer to the client computer. The set of responses is responsive to the subset of the data requests. Each of the responses employs a protocol associated with a respective one of the subset of the data requests. The method also includes receiving at the client computer at least a subset of the responses. There is further included selecting, for the communication between the client computer and the server computer, the most advantageous protocol from protocols associated with the subset of the responses, wherein the most advantageous protocol is determined based on a predefined protocol priority.

In yet another embodiment, the invention relates to a computer readable medium containing computer-readable instructions for automatically detecting a most advantageous protocol for communication by a client computer. The client computer is configured to be coupled to a server computer via a computer network. The computer-readable instructions comprise computer readable instructions for sending in a substantially parallel manner, from the client computer to the server computer, a plurality of data requests. Each of the data requests employs a different protocol and a different connection. The data requests are configured to solicit, responsive to the data requests, a set of responses from the server computer. Each of the responses employs a protocol associated with a respective one of the data requests.

The computer readable medium further includes computer readable instructions for receiving at the client computer at least a subset of the responses. There is further included computer readable instructions for monitoring the subset of the responses as each response is received from the server computer to select the most advantageous protocol from protocols associated with the subset of the responses, wherein the most advantageous protocol is determined based on a predefined protocol priority.

These and other features of the present invention will be described in more detail below in the detailed description of the invention and in conjunction with the following figures.

BRIEF DESCRIPTION OF THE DRAWINGS

To facilitate discussion, FIG. 1 illustrates a computer network, representing for example a portion of an international computer network popularly known as the Internet.

5

FIG. 2 is a block diagram of an exemplar computer system for carrying out the autodetect technique according to one embodiment of the invention.

FIG. 3 illustrates, in accordance with one embodiment, the control and data connections between a client application and a server computer when no firewall is provided in the network.

FIG. 4 illustrates another network arrangement wherein control and data connections are established through a firewall.

FIGS. 5A-B illustrates another network arrangement wherein multiple HTTP control commands and media data may be communicated between a client computer and a server computer using the HTTP protocol.

FIGS. 5C-D illustrate another network arrangement wherein multiple HTTP control and data connections are multiplexed through a single HTTP port.

FIG. 6 illustrates another network arrangement wherein control and data connections are transmitted between the client application and the server computer via a proxy.

FIG. 7 depicts, in accordance with one embodiment of the present invention, a simplified flowchart illustrating the steps of the inventive autodetect technique.

FIG. 8A depicts, in accordance with one aspect of the present invention, the steps involved in executing the UDP protocol thread of FIG. 7.

FIG. 8B depicts, in accordance with one aspect of the present invention, the steps involved in executing the TCP protocol thread of FIG. 7.

FIG. 8C depicts, in accordance with one aspect of the present invention, the steps involved in executing the HTTP protocol thread of FIG. 7.

FIG. 8D depicts, in accordance with one aspect of the present invention, the steps involved in executing the HTTP 80 protocol thread of FIG. 7.

FIG. 8E depicts, in accordance with one aspect of the present invention, the steps involved in executing the HTTP 8080 protocol thread of FIG. 7.

FIG. 9 illustrates, in accordance with one embodiment of the present invention, the steps involved in executing the control thread of FIG. 7.

DETAILED DESCRIPTION OF THE PREFERRED EMBODIMENTS

The present invention will now be described in detail with reference to a few preferred embodiments thereof as illustrated in the accompanying drawings. In the following description, numerous specific details are set forth in order to provide a thorough understanding of the present invention. It will be apparent, however, to one skilled in the art, that the present invention may be practiced without some or all of these specific details. In other instances, well known process steps have not been described in detail in order to not unnecessarily obscure the present invention.

In accordance with one aspect of the present invention, the client computer in a heterogeneous client-server computer network (e.g., client computer 106 in FIG. 1) is provided with an autodetect mechanism. When executed, the autodetect mechanism advantageously permits client computer 106 to select, in an efficient and automatic manner, the most advantageous protocol for communication between the client computer and its server. Once the most advantageous protocol is selected, parameters pertaining to the selected protocol are saved to enable the client computer, in future sessions, to employ the same selected protocol for communication.

6

In accordance with one particular advantageous embodiment, the inventive autodetect mechanism simultaneously employs multiple threads, through multiple connections, to initiate communication with the server computer, e.g., server 104. Each thread preferably employs a different protocol and requests the server computer to respond to the client computer using the protocol associated with that thread. For example, client computer 106 may, employing the autodetect mechanism, initiate five different threads, using respectively the TCP, UDP, one of HTTP and HTTP proxy, HTTP through port (multiplex) 80, and HTTP through port (multiplex) 8080 protocols to request server 104 to respond.

Upon receiving a request, server 104 responds with data using the same protocol as that associated with the thread on which the request arrives. If one or more protocols is blocked and fails to reach server 104 (e.g., by a firewall), no response employing the blocked protocol would of course be transmitted from server 104 to client computer 106. Further, some of the protocols transmitted from server 104 to client computer 106 may be blocked as well. Accordingly, client computer may receive only a subset of the responses sent from server 104.

In one embodiment, client computer 106 monitors the set of received responses. If the predefined "best" protocol is received, that protocol is then selected for communication by client computer 106. The predefined "best" protocol may be defined in advance by the user and/or the application program. If the predefined "best" protocol is, however, blocked (as the request is transmitted from the client computer or as the response is transmitted from the server, for example) the most advantageous protocol may simply be selected from the set of protocols received back by the client computer. In one embodiment, the selection may be made among the set of protocols received back by the client computer within a predefined time period after the requests are sent out in parallel.

The selection of the most advantageous protocol for communication among the protocols received by client computer 106 may be performed in accordance with some predefined priority. For example, in the real-time data rendering application, the UDP protocol may be preferred over TCP protocol, which may be in turned preferred over the HTTP protocol. This is because UDP protocol typically can handle a greater data transmission rate and may allow client computer 106 to exercise a greater degree of control over the transmission of data packets.

HTTP data, while popular nowadays for use in transmitting web pages, typically involves a higher number of overhead bits, making it less efficient relative to the UDP protocol for transmitting real-time data. As is known, the HTTP protocol is typically built on top of TCP. The underlying TCP protocol typically handles the transmission and retransmission requests of individual data packets automatically. Accordingly, the HTTP protocol tends to reduce the degree of control client computer 106 has over the transmission of the data packets between server 104 and client computer 106. Of course other priority schemes may exist for different applications, or even for different real-time data rendering applications.

In one embodiment, as client computer 106 is installed and initiated for communication with server 104 for the first time, the autodetect mechanism is invoked to allow client computer 106 to send transmission requests in parallel (e.g., using different protocols over different connections) in the manner discussed earlier. After server 104 responds with

data via multiple connections/protocols and the most advantageous protocol has been selected by client computer 106 for communication (in accordance with some predefined priority), the parameters associated with the selected protocol are then saved for future communication.

Once the most advantageous protocol is selected, the autodetect mechanism may be disabled, and future communication between client computer 106 and server 104 may proceed using the selected most advantageous protocol without further invocation of the autodetect mechanism. If the topology of computer network 100 changes and communication using the previously selected "most advantageous" protocol is no longer appropriate, the autodetect mechanism may be executed again to allow client computer 106 to ascertain a new "most advantageous" protocol for communication with server 104. In one embodiment, the user of client computer 106 may, if desired, initiate the autodetect mechanism at anytime in order to enable client computer 106 to update the "most advantageous" protocol for communication with server 104 (e.g., when the user of client computer 106 has reasons to suspect that the previously selected "most advantageous" protocol is no longer the most optimal protocol for communication).

The inventive autodetect mechanism may be implemented either in software or hardware, e.g., via an IC chip. If implemented in software, it may be carried out by any number of computers capable of functioning as a client computer in a computer network. FIG. 2 is a block diagram of an exemplar computer system 200 for carrying out the autodetect technique according to one embodiment of the invention. Computer system 200, or an analogous one, may be employed to implement either a client or a server of a computer network. The computer system 200 includes a digital computer 202, a display screen (or monitor) 204, a printer 206, a floppy disk drive 208, a hard disk drive 210, a network interface 212, and a keyboard 214. The digital computer 202 includes a microprocessor 216, a memory bus 218, random access memory (RAM) 220, read only memory (ROM) 222, a peripheral bus 224, and a keyboard controller 226. The digital computer 200 can be a personal computer (such as an Apple computer, e.g., an Apple Macintosh, an IBM personal computer, or one of the compatibles thereof), a workstation computer (such as a Sun Microsystems or Hewlett-Packard workstation), or some other type of computer.

The microprocessor 216 is a general purpose digital processor which controls the operation of the computer system 200. The microprocessor 216 can be a single-chip processor or can be implemented with multiple components. Using instructions retrieved from memory, the microprocessor 216 controls the reception and manipulation of input data and the output and display of data on output devices.

The memory bus 218 is used by the microprocessor 216 to access the RAM 220 and the ROM 222. The RAM 220 is used by the microprocessor 216 as a general storage area and as scratch-pad memory, and can also be used to store input data and processed data. The ROM 222 can be used to store instructions or program code followed by the microprocessor 216 as well as other data.

The peripheral bus 224 is used to access the input, output, and storage devices used by the digital computer 202. In the described embodiment, these devices include the display screen 204, the printer device 206, the floppy disk drive 208, the hard disk drive 210, and the network interface 212, which is employed to connect computer 200 to the network. The keyboard controller 226 is used to receive input from

keyboard 214 and send decoded symbols for each pressed key to microprocessor 216 over bus 228.

The display screen 204 is an output device that displays images of data provided by the microprocessor 216 via the peripheral bus 224 or provided by other components in the computer system 200. The printer device 206 when operating as a printer provides an image on a sheet of paper or a similar surface. Other output devices such as a plotter, typesetter, etc. can be used in place of, or in addition to, the printer device 206.

The floppy disk drive 208 and the hard disk drive 210 can be used to store various types of data. The floppy disk drive 208 facilitates transporting such data to other computer systems, and hard disk drive 210 permits fast access to large amounts of stored data.

The microprocessor 216 together with an operating system operate to execute computer code and produce and use data. The computer code and data may reside on the RAM 220, the ROM 222, the hard disk drive 220, or even on another computer on the network. The computer code and data could also reside on a removable program medium and loaded or installed onto the computer system 200 when needed. Removable program mediums include, for example, CD-ROM, PC-CARD, floppy disk and magnetic tape.

The network interface circuit 212 is used to send and receive data over a network connected to other computer systems. An interface card or similar device and appropriate software implemented by the microprocessor 216 can be used to connect the computer system 200 to an existing network and transfer data according to standard protocols.

The keyboard 214 is used by a user to input commands and other instructions to the computer system 200. Other types of user input devices can also be used in conjunction with the present invention. For example, pointing devices such as a computer mouse, a track ball, a stylus, or a tablet can be used to manipulate a pointer on a screen of a general-purpose computer.

The invention can also be embodied as computer readable code on a computer readable medium. The computer readable medium is any data storage device that can store data which can be thereafter be read by a computer system. Examples of the computer readable medium include read-only memory, random-access memory, CD-ROMs, magnetic tape, optical data storage devices. The computer readable code can also be distributed over a network coupled computer systems so that the computer readable code is stored and executed in a distributed fashion.

FIGS. 3-6 below illustrate, to facilitate discussion, some possible arrangements for the transmission and receipt of data in a computer network. The arrangements differ depend on which protocol is employed and the configuration of the network itself. FIG. 3 illustrates, in accordance with one embodiment, the control and data connections between a client application 300 and server 302 when no firewall is provided in the network.

Client application 300 may represent, for example, the executable codes for executing a real-time data rendering program such as the Web Theater Client 2.0, available from Vxtreme, Inc. of Sunnyvale, Calif. In the example of FIG. 3, client application 300 includes the inventive autodetect mechanism and may represent a plug-in software module that may be installed onto a browser 306. Browser 306 may represent, for example, the application program which the user of the client computer employs to navigate the network. By way of example, browser 306 may represent one of the popular Internet browser programs, such as Netscape™ by

Netscape Communications Inc. of Mountain View, Calif. or Microsoft Explorer by Microsoft Corporation of Redmond, Wash.

When the autodetect mechanism of client application 300 is executed in browser 306 (e.g., during the set up of client application 300), client application 300 sends a control request over control connection 308 to server 302. Although multiple control requests are typically sent in parallel over multiple control connections using different protocols as discussed earlier, only one control request is depicted in FIG. 3 to facilitate ease of illustration.

The protocol employed to send the control request over control connection 308 may represent, for example, TCP, or HTTP. If UDP protocol is requested from the server, the request from the client may be sent via the control connection using for example the TCP protocol. Initially, each control request from client application 300 may include, for example, the server name that identifies server 302, the port through which control connection may be established, and the name of the video stream requested by client application 300. Server 302 then responds with data via data connection 310.

In FIG. 3, it is assumed that no proxies and/or firewalls exist. Accordingly, server 302 responds using the same protocol as that employed in the request. If the request employs TCP, however, server 302 may attempt to respond using either UDP or TCP data connections (depending on the specifics of the request). The response is sent to client application via data connection 310. If the protocol received by the client application is subsequently selected to be the "most advantageous" protocol, subsequent communication between client application 300 and server 302 may take place via control connection 308 and data connection 310. Subsequent control requests sent by client application 300 via control connection 308 may include, for example, stop, play, fast forward, rewind, pause, unplay, and the like. These control requests may be utilized by server 302 to control the delivery of the data stream from server 302 to client application 300 via data connection 310.

It should be noted that although only one control connection and one data connection is shown in FIG. 3 to simplify illustration, multiple control and data connections utilizing the same protocol may exist during a data rendering session. Multiple control and data connections may be required to handle the multiple data streams (e.g., audio, video, annotation) that may be needed in a particular data rendering session. If desired, multiple clients applications 300 may be installed within browser 306, e.g., to simultaneously render multiple video clips, each with its own sound and annotations.

FIG. 4 illustrates another network arrangement wherein control and data connections are established through a firewall. As mentioned earlier, a firewall may have policies that restrict or prohibit the traversal of certain types of data and/or protocols. In FIG. 4, a firewall 400 is disposed between client application 300 and server 402. Upon execution, client application 300 sends control request using a given protocol via firewall 400 to server 402. Server 402 then responds with data via data connection 410, again via firewall 400.

If the data and/or protocol can be received by the client computer through firewall 400, client application 300 may then receive data from server 402 (through data connection 408) in the same protocol used in the request. As before, if the request employs the TCP protocol, the server may respond with data connections for either TCP or UDP

protocol (depending on the specifics of the request). Protocols that may traverse a firewall may include one or more of the following: UDP, TCP, and HTTP.

In accordance with one aspect of the present invention, the HTTP protocol may be employed to send/receive media data (video, audio, annotation, or the like) between the client and the server. FIG. 5A is a prior art drawing illustrating how a client browser may communicate with a web server using a port designated for communication. In FIG. 5, there is shown a web server 550, representing the software module for serving web pages to a browser application 552. Web server 550 may be any of the commercially available web servers that are available from, for example, Netscape Communications Inc. of Mountain View, Calif. or Microsoft Corporation of Redmond, Wash. Browser application 552 represents for example the Netscape browser from the aforementioned Netscape Communications, Inc., or similarly suitable browser applications.

Through browser application 552, the user may, for example, obtain web pages pertaining to a particular entity by sending an HTTP request (e.g., GET) containing the URL (uniform resource locator) that identifies the web page file. The request sent via control connection 553 may arrive at web server 550 through the HTTP port 554. HTTP port 554 may represent any port through which HTTP communication is enabled. HTTP port 554 may also represent the default port for communicating web pages with client browsers. The HTTP default port may represent, for example, either port 80 or port 8080 on web server 550. As is known, one or both of these ports on web server 550 may be available for web page communication even if there are firewalls disposed between the web server 550 and client browser application 552, which otherwise block all HTTP traffic in other ports. Using the furnished URL, web server 550 may then obtain the desired web page(s) for sending to client browser application 552 via data connection 556.

The invention, in one embodiment, involves employing the HTTP protocol to communicate media commands from a browser application or browser plug-in to the server. Media commands are, for example, PLAY, STOP, REWIND, FAST FORWARD, and PAUSE. The server computer may represent, for example, a web server. The server computer may also represent a video server for streaming video to the client computer. Through the use of the HTTP protocol the client computer may successfully send media control requests and receive media data through any HTTP port. If the default HTTP port, e.g., port 80 or 8080, is specified, the client may successfully send media control requests and receive media data even if there exists a firewall or an HTTP Proxy disposed in between the server computer and the client computer, which otherwise blocks all other traffic that does not use the HTTP protocol. For example, these firewalls or HTTP Proxies do not allow regular TCP or UDP packets to go through.

As is well known to those skilled, the HTTP protocol, as specified by for example the Internet Request For Comments RFC 1945 (T. Berners-Lee et al.), typically defines only three types of requests to be sent from the client computer to the server, namely GET, POST, and HEAD. The POST command, for instance, is specified in RFC 1945 to be composed of a Request-Line, one or more Headers and Entity-Body. To send media commands like PLAY, REWIND, etc., the invention in one embodiment sends the media command as part of the Entity-Body of the HTTP POST command. The media command can be in any format or protocol, and can be, for instance, in the same format as that employed when firewalls are not a concern and plain

TCP protocol can be used. This format can be, for example, RTSP (Real Time Streaming Protocol).

When a server gets an HTTP request, it answers the client with an HTTP Response. Responses are typically composed of a Status-Line, one or more headers, and an Entity-Body. In one embodiment of this invention, the response to the media commands is sent as the Entity-Body of the response to the original HTTP request that carried the media command.

FIG. 5B illustrates this use of HTTP for sending arbitrary media commands. In FIG. 5B, the plug-in application 560 within client browser application 562 may attempt to receive media data (e.g., video, audio, annotation, or the like) by first sending an HTTP request to server 564 via control connection 565. For example, a REWIND command could be sent from the client 560 to the server 564 as an HTTP packet 570 of the form: "POST/HTTP/1.0<Entity-Body containing rewind command in any suitable media protocol>". The server can answer to this request with an HTTP response 572 of the form: "HTTP/1.0 200ok<Entity-Body containing rewind response in any suitable media protocol>".

The HTTP protocol can be also used to send media data across firewalls. The client can send a GET request to the video server, and the video server can then send the video data as the Entity-Body of the HTTP response to this GET request.

Some firewalls may be restrictive with respect to HTTP data and may permit HTTP packets to traverse only on a certain port, e.g., port 80 and/or port 8080. FIG. 5C illustrates one such situation. In this case, the control and data communications for the various data stream, e.g., audio, video, and/or annotation associated with different rendering sessions (and different clients) may be multiplexed using conventional multiplexer code and/or circuit 506 at client application 300 prior to being sent via port 502 (which may represent, for example, HTTP port 80 or HTTP port 8080). The inventive combined use of the HTTP protocol and of the multiplexer for transmitting media control and data is referred to as the HTTP multiplex protocol, and can be used to send this data across firewalls that only allow HTTP traffic on specific ports, e.g., port 80 or 8080.

At server 402, representing, for example, server 104 of FIG. 1, conventional demultiplexer code and/or circuit 508 may be employed to decode the received data packets to identify which stream the control request is associated with. Likewise, data sent from server 402 to client application 300 may be multiplexed in advance at server 402 using for example conventional multiplexer code and/or circuit 510. The multiplexed data is then sent via port 502. At client application 300, the multiplexed data may be decoded via conventional demultiplexer code and/or circuit 512 to identify which stream the received data packets is associated with (audio, video, or annotation).

Multiplexing and demultiplexing at the client and/or server may be facilitated for example by the use of the Request-URL part of the Request-Line of HTTP requests. As mentioned above, the structure of HTTP requests is described in RFC 1945. The Request-URL may, for example, identify the stream associated with the data and/or control request being transmitted. In one embodiment, the additional information in the Request-URL in the HTTP header may be as small as one or a few bits added to the HTTP request sent from client application 300 to server 402.

To further facilitate discussion of the inventive HTTP multiplexing technique, reference may now be made to FIG. 5D. In FIG. 5D, the plug-in application 660 within client

plug-in application 660 may attempt to receive media data (e.g., video, audio, annotation, or the like) by first sending a control request 670 to server 664 via control connection 665. The control request is an HTTP request, which arrives at the HTTP default port 654 on server 664. As mentioned earlier, the default HTTP port may be either port 80 or port 8080 in one embodiment.

In one example, the control request 670 from client plug-in 660 takes the form of a command to "POST/12469 HTTP/1.0<Entity-Body>" which indicates to the server (through the designation 12469 as the Request-URL) that this is a control connection. The Entity-Body contains, as described above, binary data that informs the video server that the client plug-in 660 wants to display a certain video or audio clip. Software codes within server 664 may be employed to assign a unique ID to this particular request from this particular client.

For discussion sake, assume that server 664 associates unique ID 35,122 with a video data connection between itself and client plug-in application 660, and unique ID 29,999 with an audio data connection between itself and client plug-in application. The unique ID is then communicated as message 672 from server 664 to client plug-in application 660, again through the aforementioned HTTP default port using data connection 667. The Entity-Body of message 672 contains, among other things and as depicted in detail 673, the audio and/or video session ID. Note that the unique ID is unique to each data connection (e.g., each of the audio, video, and annotation connections) of each client plug-in application (since there may be multiple client plug-in applications attempting to communicate through the same port).

Once the connection is established, the same unique ID number is employed by the client to issue HTTP control requests to server 664. By way of example, client plug-in application 660 may issue a command "GET /35,122 HTTP/1.0" or "POST /35,122 HTTP/1.0<Entity-Body containing binary data with the REWIND media command>" to request a video file or to rewind on the video file. Although the rewind command is used in FIGS. 5A-5D to facilitate ease of discussion, other media commands, e.g., fast forward, pause, real-time play, live-play, or the like, may of course be sent in the Entity-Body. Note that the unique ID is employed in place of or in addition to the Request-URL to qualify the Request-URL.

Once the command is received by server 664, the unique ID number (e.g. 35,122) may be employed by the server to demultiplex the command to associate the command with a particular client and data file. This unique ID number can also attach to the HTTP header of HTTP responses sent from server 664 to client plug-in application 660, through the same HTTP default port 654 on server 664, to permit client plug-in application 660 to ascertain whether an HTTP data packet is associated with a given data stream.

Advantageously, the invention permits media control commands and media data to be communicated between the client computer and the server computer via the default HTTP port, e.g., port 80 or 8080 in one embodiment, even if HTTP packets are otherwise blocked by a firewall disposed between the client computer and the server computer. The association of each control connection and data connection to each client with a unique ID advantageously permits multiple control and data connections (from one or more clients) to be established through the same default HTTP port on the server, advantageously bypassing the firewall. Since both the server and the client have the

demultiplexer code and/or circuit that resolve a particular unique ID into a particular data stream, multiplexed data communication is advantageously facilitated thereby.

In some networks, it may not be possible to traverse the firewall due to stringent firewall policies. As mentioned earlier, it may be possible in these situations to allow the client application to communicate with a server using a proxy. FIG. 6 illustrates this situation wherein client application 300 employs proxy 602 to communicate with server 402. The use of proxy 602 may be necessary since client application 300 may employ a protocol which is strictly prohibited by firewall 604. The identity of proxy 602 may be found in browser program 306, e.g., Netscape as it employs the proxy to download its web pages, or may be configured by the user himself. Typical protocols that may employ a proxy for communication, e.g., proxy 602, includes HTTP and UDP.

In accordance with one embodiment of the present invention, the multiple protocols that may be employed for communication between a server computer and a client computer are tried in parallel during autodetect. In other words, the connections depicted in FIGS. 3, 4, 5C, and 6 may be attempted simultaneously and in parallel over different control connections by the client computer. Via these control connections, the server is requested to respond with various protocols.

If the predefined "best" protocol (predetermined in accordance with some predefined protocol priority) is received by the client application from the server, autodetect may, in one embodiment, end immediately and the "best" protocol is selected for immediate communication. In one real-time data rendering application, UDP is considered the "best" protocol, and the receipt of UDP data by the client may trigger the termination of the autodetect.

If the "best" protocol has not been received after a predefined time period, the most advantageous protocol (in terms of for example data transfer rate and/or transmission control) is selected among the set of protocols received by the client. The selected protocol may then be employed for communication between the client and the server.

FIG. 7 depicts, in accordance with one embodiment of the present invention, a simplified flowchart illustrating the steps of the inventive autodetect technique. In FIG. 7, the client application starts (in step 702) by looking up the HTTP proxy, if there is any, from the browser. As stated earlier, the client computer may have received a web page from the browser, which implies that the HTTP protocol may have been employed by the browser program for communication. If a HTTP proxy is required, the name and location of the HTTP proxy is likely known to the browser, and this knowledge may be subsequently employed by the client to at least enable communication with the server using the HTTP proxy protocol, i.e., if a more advantageous protocol cannot be ascertained after autodetect.

In step 704, the client begins the autodetect sequence by starting in parallel the control thread 794, along with five protocol threads 790, 792, 796, 798, and 788. As the term is used herein, parallel refers to both the situation wherein the multiple protocol threads are sent parallelly starting at substantially the same time (having substantially similar starting time), and the situation wherein the multiple protocol threads simultaneously execute (executing at the same time), irrespective when each protocol thread is initiated. In the latter case, the multiple threads may have, for example, staggered start time and the initiation of one thread may not depend on the termination of another thread.

Control thread 794 represents the thread for selecting the most advantageous protocol for communication. The other protocol threads 790, 792, 796, 798, and 788 represent threads for initiating in parallel communication using the various protocols, e.g., UDP, TCP, HTTP proxy, HTTP through port 80 (HTTP 80), and HTTP through port 8080 (HTTP 8080). Although only five protocol threads are shown, any number of protocol threads may be initiated by the client, using any conventional and/or suitable protocols. The steps associated with each of threads 794, 790, 792, 796, 798, and 788 are discussed herein in connection with FIGS. 8A-8E and 9.

In FIG. 8A, the UDP protocol thread is executed. The client inquires in step 716 whether there requires a UDP proxy. If the UDP proxy is required, the user may obtain the name of the UDP proxy from, for example, the system administrator in order to use the UDP proxy to facilitate communication to the proxy (in step 718). If no UDP proxy is required, the client may directly connect to the server (in step 720). Thereafter, the client may begin sending a data request (i.e., a control request) to the server in step 722 using the UDP protocol (either through the proxy if a proxy is involved or directly to the server if no proxy is required).

In FIG. 8B, the TCP protocol thread is executed. If TCP protocol is employed, the client typically directly connects to the server (in step 726). Thereafter, the client may begin sending a data request (i.e., a control request) to the server using the TCP protocol (step 724).

In FIG. 8C, the HTTP protocol thread is executed. The client inquires in step 716 whether there requires a HTTP proxy. If the HTTP proxy is required, the user may obtain the name of the HTTP proxy from, for example, the browser since, as discussed earlier, the data pertaining to the proxy may be kept by the browser. Alternatively, the user may obtain data pertaining to the HTTP proxy from the system administrator in order to use the HTTP proxy to facilitate communication to the server (in step 732).

If no HTTP proxy is required, the client may directly connect to the server (in step 730). Thereafter, the client may begin sending a data request (i.e., a control request) to the server in step 734 using the HTTP protocol (either through the proxy if a proxy is involved or directly to the server if no proxy is required).

In FIG. 8D, the HTTP 80 protocol thread is executed. If HTTP 80 protocol is employed, HTTP data may be exchanged but only through port 80, which may be for example the port on the client computer through which communication with the network is permitted. Through port 80, the client typically directly connects to the server (in step 736). Thereafter, the client may begin sending a data request (i.e., a control request) to the server (step 738) using the HTTP 80 protocol.

In FIG. 8E, the HTTP 8080 protocol thread is executed. If HTTP 8080 protocol is employed, HTTP data may be exchanged but only through port 8080, which may be the port on the client computer for communicating with the network. Through port 8080, the client typically directly connects to the server (in step 740). Thereafter, the client may begin sending a data request (i.e., a control request) to the server (step 742) using the HTTP 8080 protocol. The multiplexing and demultiplexing techniques that may be employed for communication through port 8080, as well as port 80 of FIG. 8D, have been discussed earlier and are not repeated here for brevity sake.

FIG. 9 illustrates, in accordance with one embodiment of the present invention, control thread 794 of FIG. 7. It should

15

be emphasized that FIG. 7 is but one way of implementing the control thread; other techniques of implementing the control thread to facilitate autodetect should be apparent to those skilled in the art in view of this disclosure. In step 746, the thread determines whether the predefined timeout period has expired. The predefined timeout period may be any predefined duration (such as 7 seconds for example) from the time the data request is sent out to the server (e.g., step 722 of FIG. 8A). In one embodiment, each protocol thread has its own timeout period whose expiration occurs at the expiration of a predefined duration after the data request using that protocol has been sent out. When all the timeout periods associated with all the protocols have been accounted for, the timeout period for the autodetect technique is deemed expired.

If the timeout has occurred, the thread moves to step 754 wherein the most advantageous protocol among the set of protocols received back from the server is selected for communication. As mentioned, the selection of the most advantageous protocol may be performed in accordance with some predefined priority scheme, and data regarding the selected protocol may be saved for future communication sessions between this server and this client.

If no timeout has occurred, the thread proceeds to step 748 to wait for either data from the server or the expiration of the timeout period. If timeout occurs, the thread moves to step 754, which has been discussed earlier. If data is received from the server, the thread moves to step 750 to ascertain whether the protocol associated with the data received from the server is the predefined "best" protocol, e.g., in accordance with the predefined priority.

If the predefined "best" protocol (e.g., UDP in some real-time data rendering applications) is received, the thread preferably moves to step 754 to terminate the autodetect and to immediately begin using this protocol for data communication instead of waiting of the timeout expiration. Advantageously, the duration of the autodetect sequence may be substantially shorter than the predefined timeout period. In this manner, rapid autodetect of the most suitable protocol and rapid establishment of communication are advantageously facilitated.

If the predefined "best" protocol is not received in step 750, the thread proceeds to step 752 to add the received protocol to the received set. This received protocol set represents the set of protocols from which the "most advantageous" (relatively speaking) protocol is selected. The most advantageous protocol is ascertained relative to other protocols in the received protocol set irrespective whether it is the predefined "best" protocol in accordance with the predefined priority. As an example of a predefined protocol priority, UDP may be deemed to be best (i.e., the predefined best), followed by TCP, HTTP, then HTTP 80 and HTTP 8080 (the last two may be equal in priority). As mentioned earlier, the most advantageous protocol is selected from the received protocol set preferably upon the expiration of the predefined timeout period.

From step 752, the thread returns to step 746 to test whether the timeout period has expired. If not, the thread continues along the steps discussed earlier.

Note that since the invention attempts to establish communication between the client application and the server computer in parallel, the time lag between the time the autodetect mechanism begins to execute and the time when the most advantageous protocol is determined is minimal. If communication attempts have been tried in serial, for example, the user would suffer the delay associated with

16

each protocol thread in series, thereby disadvantageously lengthening the time period between communication attempt and successful establishment of communication.

The saving in time is even more dramatic in the event the network is congested or damaged. In some networks, it may take anywhere from 30 to 90 seconds before the client application realizes that an attempt to connect to the server (e.g., step 720, 726, 730, 736, or 740) has failed. If each protocol is tried in series, as is done in one embodiment, the delay may, in some cases, reach minutes before the user realizes that the network is unusable and attempts should be made at a later time.

By attempting to establish communication via the multiple protocols in parallel, network-related delays are suffered in parallel. Accordingly, the user does not have to wait for multiple attempts and failures before being able to ascertain that the network is unusable and an attempt to establish communication should be made at a later time. In one embodiment, once the user realizes that all parallel attempts to connect with the network and/or the proxies have failed, there is no need to make the user wait until the expiration of the timeout periods of each thread. In accordance with this embodiment, the user is advised to try again as soon as it is realized that parallel attempts to connect with the server have all failed. In this manner, less of the user's time is needed to establish optimal communication with a network.

While this invention has been described in terms of several preferred embodiments, there are alterations, permutations, and equivalents which fall within the scope of this invention. For example, although the invention has been described with reference with sending out protocol threads in parallel, the automatic protocol detection technique also applies when the protocol threads are sent serially. In this case, while it may take longer to select the most advantageous protocol for selection, the automatic protocol detection technique accomplishes the task without requiring any sophisticated technical knowledge on the part of the user of the client computer. The duration of the autodetect technique, even when serial autodetect is employed, may be shortened by trying the protocols in order of their desirability and ignoring less desirable protocols once a more desirable protocol is obtained. It should also be noted that there are many alternative ways of implementing the methods and apparatuses of the present invention. It is therefore intended that the following appended claims be interpreted as including all such alterations, permutations, and equivalents as fall within the true spirit and scope of the present invention.

What is claimed is:

1. In a computer network, a method for automatically detecting a most advantageous protocol for communication by a client computer, said client computer being configured to be coupled to a server computer via a computer network, the method comprising:

sending, from said client computer to said server computer, a plurality of data requests substantially in parallel, each of said data requests employing a different protocol and a different connection, said data requests being configured to solicit, responsive to said data requests, a set of responses from said server computer, each of said responses employing a protocol associated with a respective one of said data requests; receiving at said client computer at least a subset of said responses; monitoring said subset of said responses as each response is received from said server computer to select said most advantageous protocol from protocols

17

associated with said subset of said responses, wherein said most advantageous protocol is determined based on a predefined protocol priority that designates a predefined best protocol; and

in response to expiration of a timeout period without receiving a response employed by the predefined best protocol, selecting the most advantageous protocol from the protocols employed by the responses received at the client computer, the selection based at least in part on at least one of a data transfer rate and transmission control characteristics, said timeout period being measured from a transmitting time of a data request.

2. The method of claim 1 wherein said plurality of data requests are sent substantially at the same time.

3. The method of claim 1 wherein said plurality of data requests execute concurrently.

4. The method of claim 1 wherein said client computer, upon receiving a response employing said predefined best protocol, immediately designates said predefined best protocol as said most advantageous protocol.

5. The method of claim 1 wherein at least one data request is sent using a multiplexed HTTP protocol.

6. The method of claim 1 wherein said client computer is configured and arranged to execute an application for rendering real-time data as said real-time data is received from said server computer.

7. The method of claim 1 wherein said computer network comprises the Internet and said predefined best protocol is UDP.

8. The method of claim 7 wherein said real-time data represents one of a video data stream, an audio data stream, and an annotation data stream.

9. In a computer network, a method for automatically detecting a most advantageous protocol for communication by a client computer, said client computer being configured to be coupled to a server computer via a computer network, the method comprising:

sending, from said client computer to said server computer, a plurality of data requests substantially in parallel, each of said data requests employing a different protocol and a different connection;

receiving at least a subset of said data requests at said server computer;

sending a set of responses from said server computer to said client computer, said set of responses being responsive to said subset of said data requests, each of said responses employing a protocol associated with a respective one of said subset of said data requests;

receiving at said client computer at least a subset of said responses; and

selecting, for said communication between said client computer and said server computer, said most advantageous protocol from protocols associated with said subset of said responses, wherein said most advantageous protocol is determined based on a predefined protocol priority that designates a predefined best protocol if a response employing the predefined best protocol is received at the client computer, and wherein, if a timeout period expires without the client computer receiving a response employing the predefined best protocol, the most advantageous protocol is selected from the protocols employed by the responses received at the client computer based at least in part on at least one of a data transfer rate and transmission control characteristics, said timeout period being measured from a transmitting time of a data request.

18

10. The method of claim 9 wherein selecting the most advantageous protocol comprises:

monitoring, employing said client computer, said subset of said responses as each one of said subset of said responses is received at said client computer from said server computer for a response employing said predefined best protocol; and

designating said predefined best protocol said most advantageous protocol, thereby immediately permitting said client computer to employ said predefined best protocol for communication with said server computer without further receiving additional responses from said server computer.

11. The method of claim 9 wherein said computer network comprises the Internet and said predefined best protocol is UDP.

12. The method of claim 9 wherein said client computer is configured and arranged to execute an application for rendering real-time data as said real-time data is received from said server computer.

13. The method of claim 12 wherein said real-time data represents one of a video data stream, an audio data stream, and an annotation data stream.

14. The method of claim 13 wherein said data requests employ at least one of a UDP, TCP, HTTP proxy, HTTP 80, and HTTP 8080 protocols.

15. The method of claim 14 wherein said HTTP 80 protocol is a protocol for permitting multiple HTTP data streams to be transmitted in a multiplexed manner through port 80, said multiple HTTP data streams representing said video data stream and said audio data stream.

16. The method of claim 14 wherein said HTTP 8080 protocol is a protocol for permitting multiple HTTP data streams to be transmitted in a multiplexed manner through port 8080, said multiple HTTP data streams representing said video data stream and said audio data stream.

17. A computer-readable medium containing computer-readable instructions for automatically detecting a most advantageous protocol for communication by a client computer, said client computer being configured for coupling to a server computer via a computer network, said computer-readable instructions comprise:

computer-readable instructions for sending in a substantially parallel manner, from said client computer to said server computer, a plurality of data requests, each of said data requests employing a different protocol and a different connection, said data requests being configured to solicit, responsive to said data requests, a set of responses from said server computer, each of said responses employing a protocol associated with a respective one of said data requests;

computer-readable instructions for receiving at said client computer at least a subset of said responses;

computer-readable instructions for monitoring said subset of said responses as each response is received from said server computer to select said most advantageous protocol from protocols associated with said subset of said responses, wherein said most advantageous protocol is determined based on a predefined protocol priority that designates a predefined best protocol; and

computer-readable instructions for, in response to expiration of a timeout period without receiving a response employing the predefined best protocol, selecting the most advantageous protocol from the protocols employed by the responses received at the client computer, the selection based at least in part on at least

19

one of a data transfer rate and transmission control characteristics, said timeout period being measured from a transmitting time of a data request.

18. The computer-readable medium of claim 17 wherein said client computer, upon receiving a response employing said predefined best protocol, immediately designates said predefined best protocol said most advantageous protocol.

19. The computer-readable medium of claim 17 wherein at least one data request is sent using a multiplexed HTTP protocol.

20. The computer-readable medium of claim 17 wherein said client computer is configured and arranged to execute an application for rendering real-time data as said real-time data is received from said server computer.

21. The computer-readable medium of claim 17 wherein said computer network comprises the Internet and said predefined best protocol is UDP.

22. The computer readable medium of claim 21 wherein said real-time data represents one of a video data stream, an audio data stream, and an annotation data stream.

23. A computer-readable medium containing computer-readable instructions for automatically detecting a most advantageous protocol for communication by a client computer, said client computer being, configured for coupling to a server computer via a computer network, said computer-readable instructions comprise:

computer-readable instructions for sending from said client computer to said server computer, a plurality of data requests substantially in parallel, each of said data requests employing a different protocol and a different connection;

20

computer-readable instructions for receiving, at least a subset of said data requests at said server computer;

computer-readable instructions for sending a set of responses from said server computer to said client computer, said set of responses being responsive to said subset of said data requests, each of said responses employing a protocol associated with a respective one of said subset of said data requests;

computer-readable instructions for receiving at said client computer at least a subset of said responses; and

computer-readable instructions for selecting, for said communication between said client computer and said server computer, said most advantageous protocol from protocols associated with said subset of said responses, wherein said most advantageous protocol is determined based on a predefined protocol priority that designates a predefined best protocol if a response employing the predefined best protocol is received at the client computer, and wherein, if a timeout period expires without the client computer receiving a response employing the predefined best protocol, the most advantageous protocol is selected from the protocols employed by the responses received at the client computer based at least in part on at least one of a data transfer rate and transmission control characteristics, said timeout period being measured from a transmitting time of a data request.

* * * * *



US005870549A

United States Patent [19]
Bobo, II[11] **Patent Number:** **5,870,549**[45] **Date of Patent:** ***Feb. 9, 1999**[54] **SYSTEMS AND METHODS FOR STORING, DELIVERING, AND MANAGING MESSAGES**[76] **Inventor:** **Charles R. Bobo, II**, 569 Elmwood Dr., NE., Atlanta, Ga. 30306[*] **Notice:** The term of this patent shall not extend beyond the expiration date of Pat. No. 5,675,507.

5,479,411	12/1995	Klein .
5,483,580	1/1996	Brandman et al. .
5,497,373	3/1996	Hulen et al. .
5,526,353	6/1996	Henley et al. .
5,608,786	3/1997	Gordon .
5,675,507	10/1997	Bobo, II .

FOREIGN PATENT DOCUMENTS

WO 96/34341 10/1996 WIPO .

OTHER PUBLICATIONS

Delrina Advertisement, 1994.

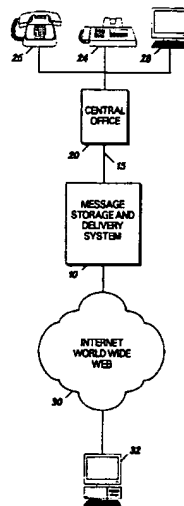
"Working with . . . Fax Mailbox" PCToday by Jim Cope (Sep. 1994, vol. 8, Issue 9).

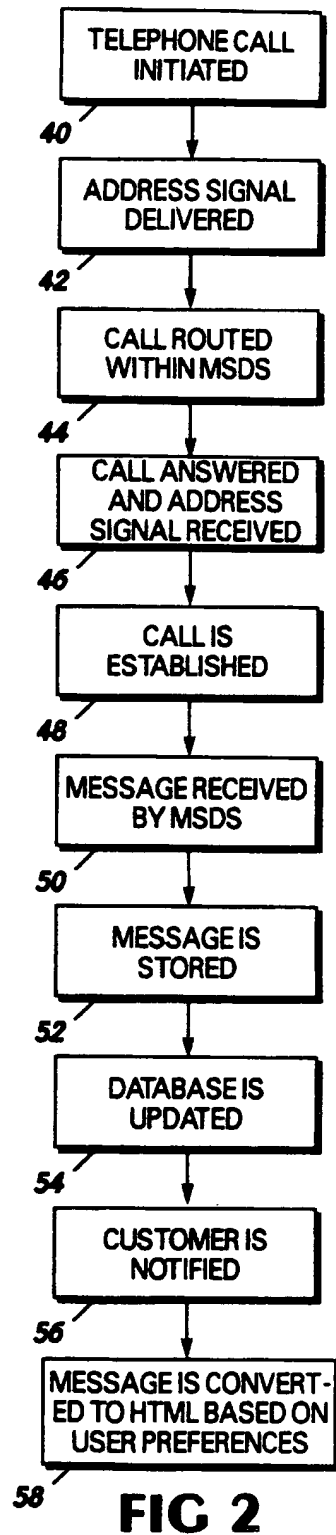
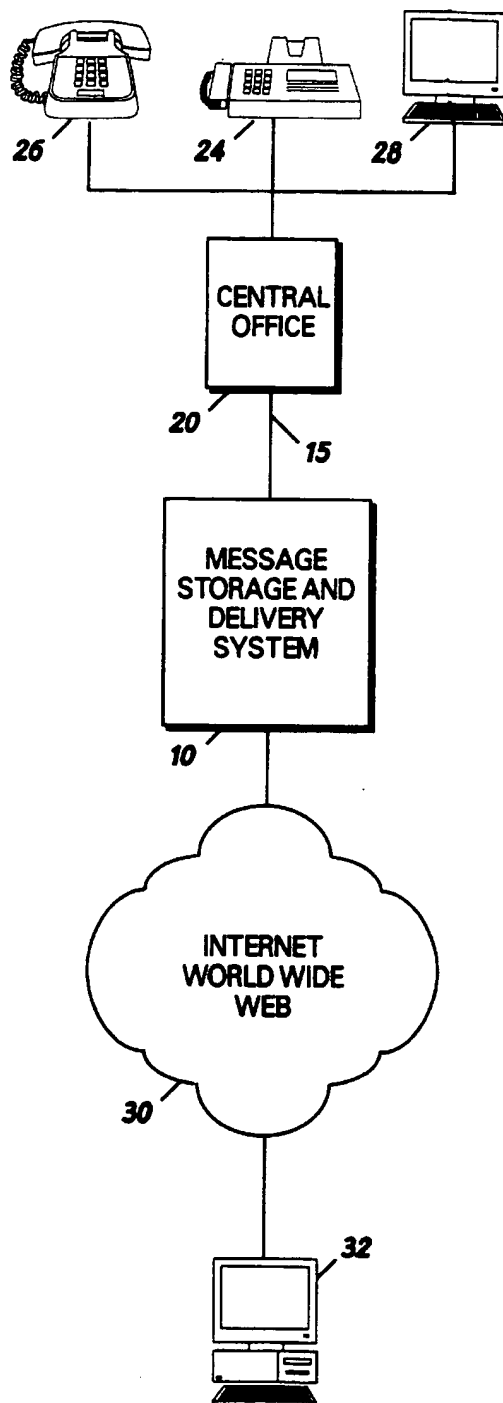
Voice/Fax Combos by Stuart Warren, *Computer Telephony*, Sep./Oct. 1994, p. 88.*Primary Examiner*—Thomas Peeso*Attorney, Agent, or Firm*—Geoff L. Sutcliffe; Kilpatrick Stockton LLP[21] **Appl. No.:** **944,741**[22] **Filed:** **Oct. 6, 1997****Related U.S. Application Data**[63] **Continuation-in-part of Ser. No. 431,716**, Apr. 28, 1995, Pat. No. 5,675,507.[51] **Int. Cl.⁶** **H04N 1/413**[52] **U.S. Cl.** **395/200.36; 348/14; 348/17; 358/400; 358/402**[58] **Field of Search** **395/200.36, 154; 348/17, 14; 358/400, 402, 403; 340/311.1**[56] **References Cited****U.S. PATENT DOCUMENTS**

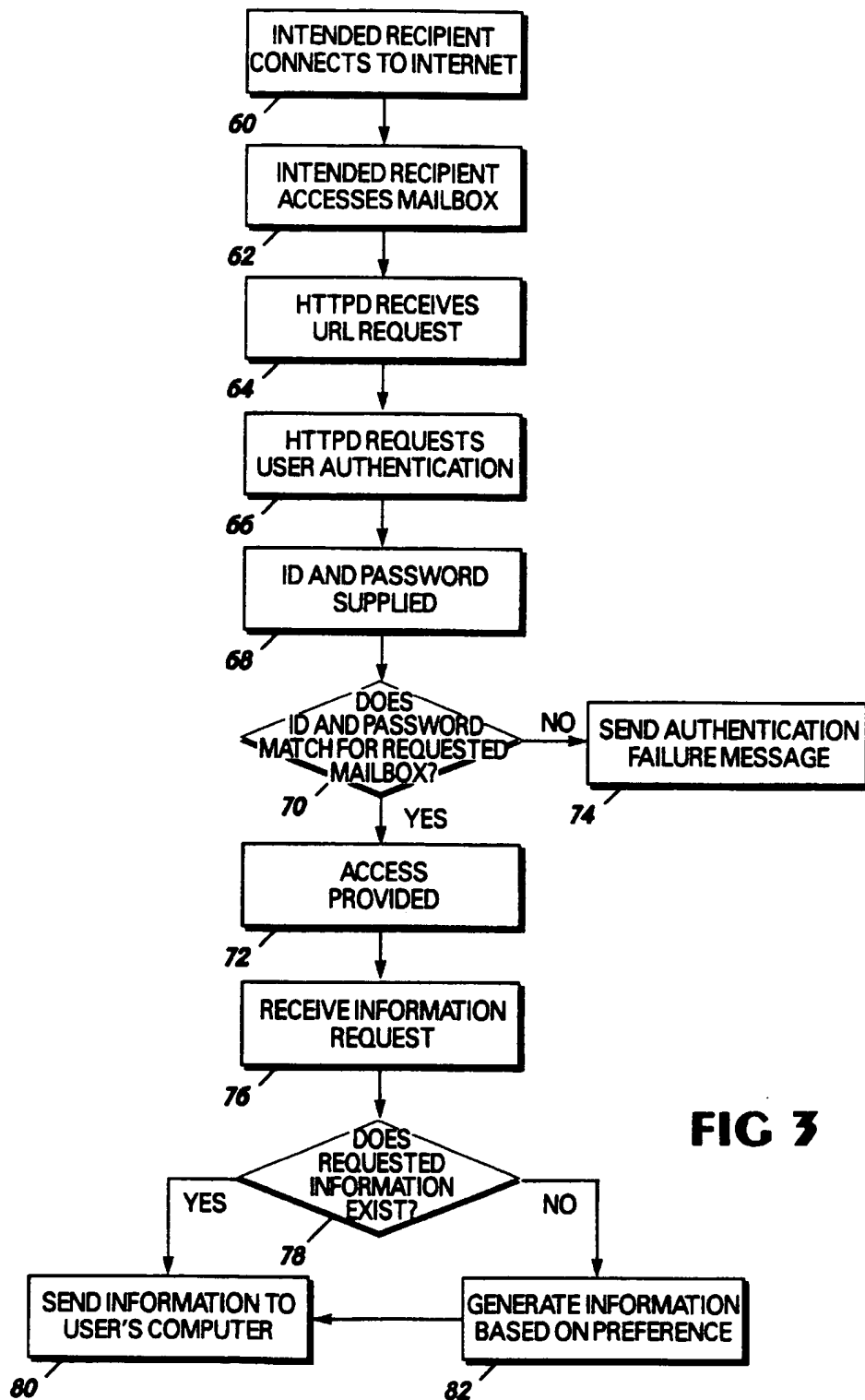
4,106,060	8/1978	Chapman, Jr. .
4,918,722	4/1990	Duehren et al. .
5,033,079	7/1991	Catron et al. .
5,065,427	11/1991	Godbole .
5,068,888	11/1991	Scherk et al. .
5,091,790	2/1992	Silverberg .
5,115,326	5/1992	Burgess et al. .
5,175,762	12/1992	Kochis et al. .
5,247,591	9/1993	Baran .
5,255,312	10/1993	Koshiishi .
5,257,112	10/1993	Okada .
5,291,302	3/1994	Gordon et al. .
5,291,546	3/1994	Giler et al. .
5,317,628	5/1994	Misholi et al. .
5,333,266	7/1994	Boaz et al. .
5,349,636	9/1994	Iribarren .

[57] **ABSTRACT**

A Message Storage and Deliver System (MSDS) is connected to the public switched telephone network (PSTN) and receives incoming calls with these calls being facsimile, voice, or data transmissions. The MSDS detects the type of call and stores the message signal in a database. The MSDS is also connected to the Internet and has a hyper-text transfer protocol daemon (HTTPD) for receiving requests from users. The HTTPD forwards requests for certain files or messages to a network server which transmits at least part of the message to the HTTPD and then to the user. In addition to requests for certain documents, the HTTPD may also receive a request in the form of a search query. The search query is forwarded from the HTTPD to an application program for conducting the search of the database. The results of the search are forwarded through the HTTPD to the user. The user may then select one or more files or messages from the search results and may save the search for later reference.

4 Claims, 18 Drawing Sheets





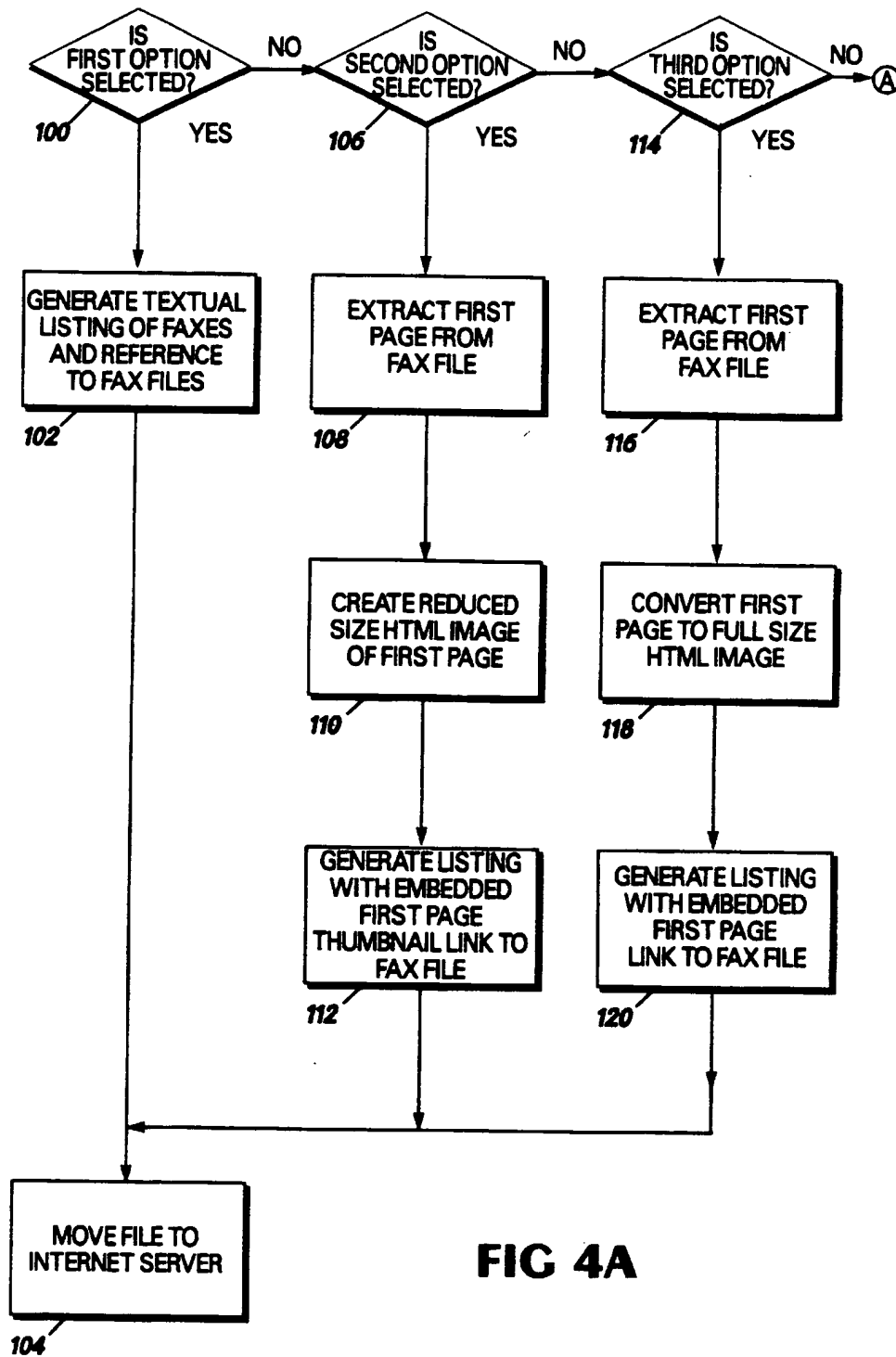
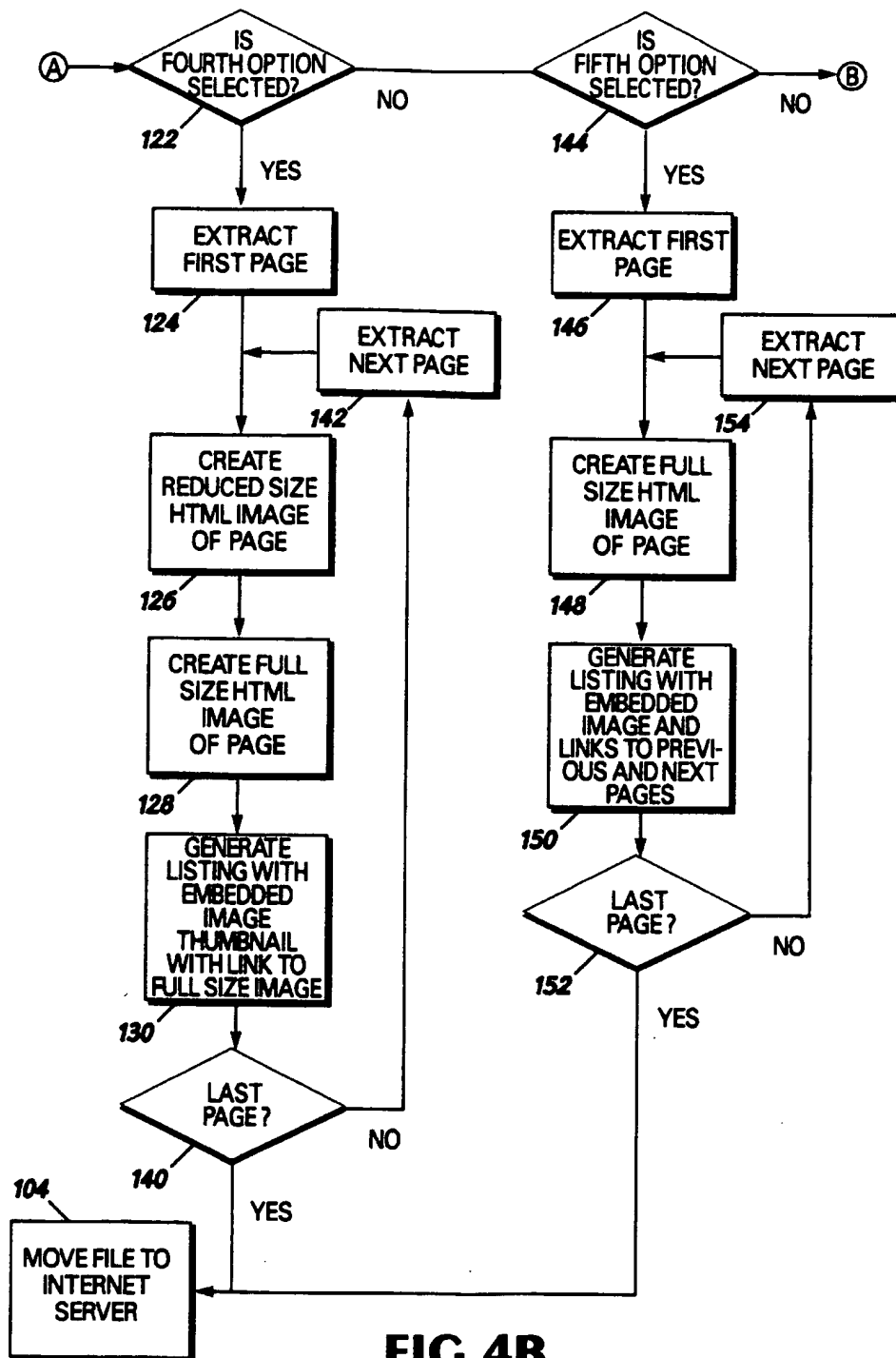
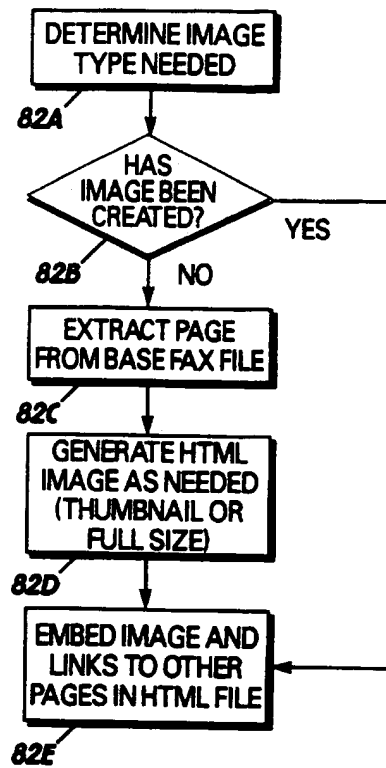
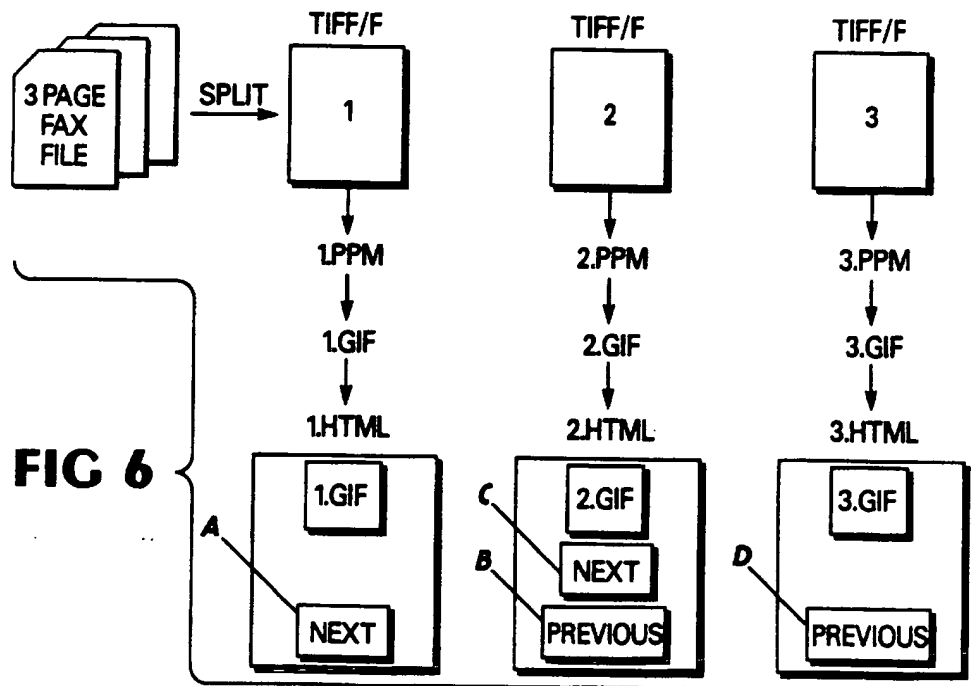


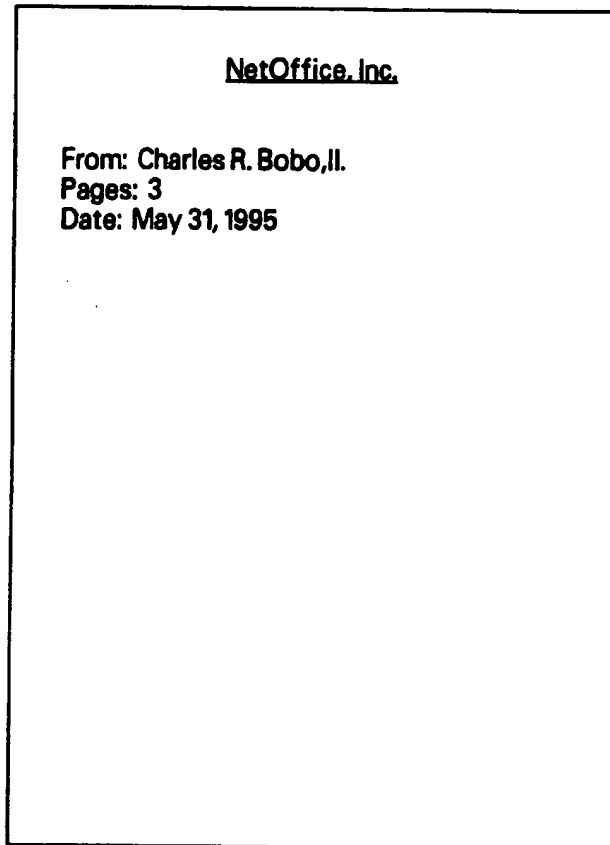
FIG 4A

**FIG 4B**

**FIG 5****FIG 6**

Fax from (404)249-6801

Received on May 31, 1995 at 1:58 PM
Page 1 of 3

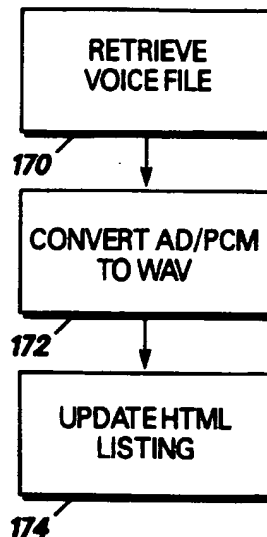
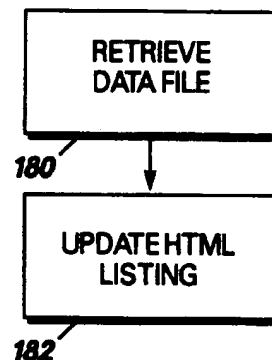


Next Page

Return to Fax Listing

This page was automatically generated by FaxWeb(tm) On May 31, 1995 at 2:05pm.
©1995 NetOffice, inc.

NetOffice, inc.
PO Box 7115
Atlanta, GA 30357
info@netoffice.com

FIG 7**FIG 8****FIG 9**

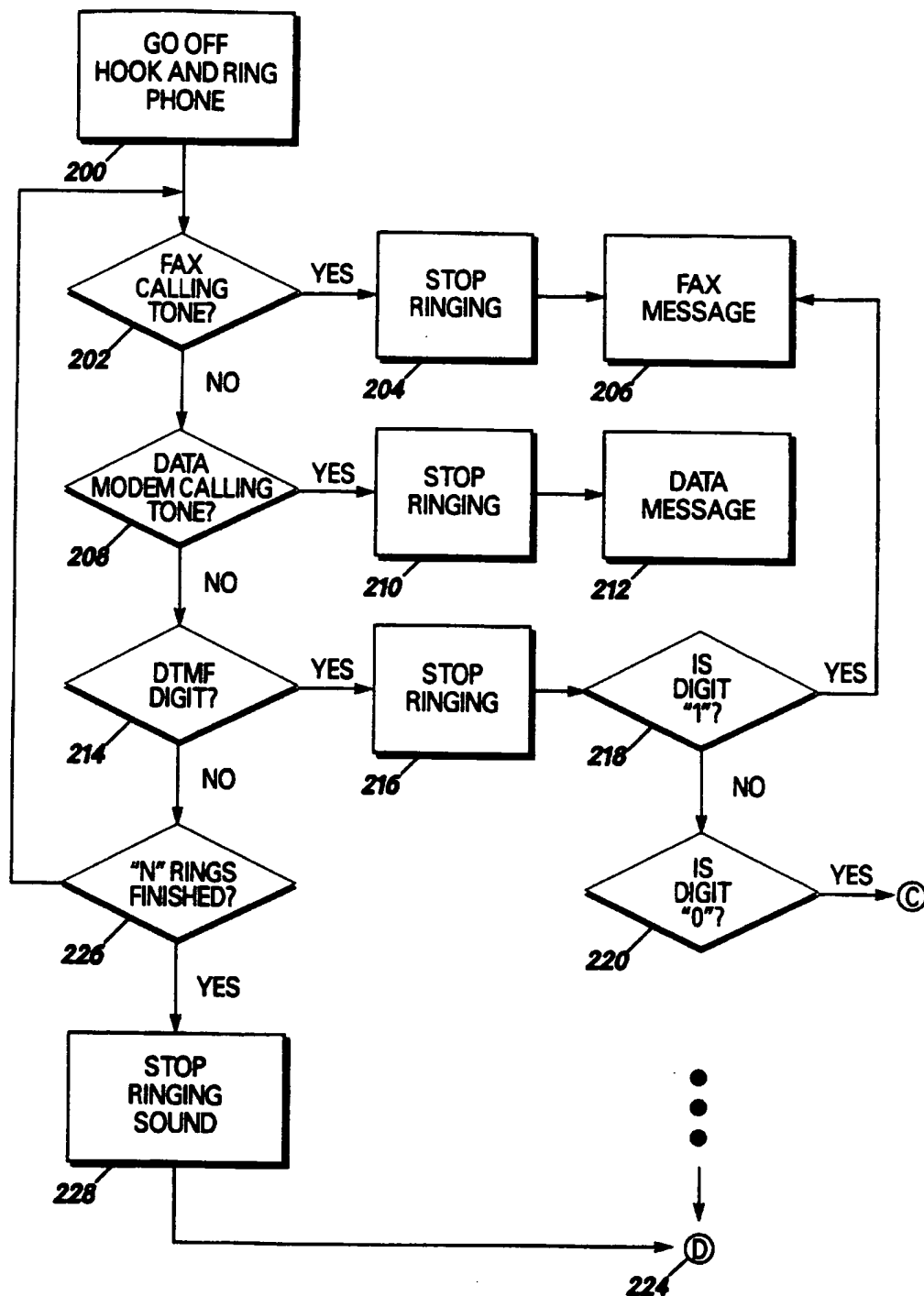


FIG 10

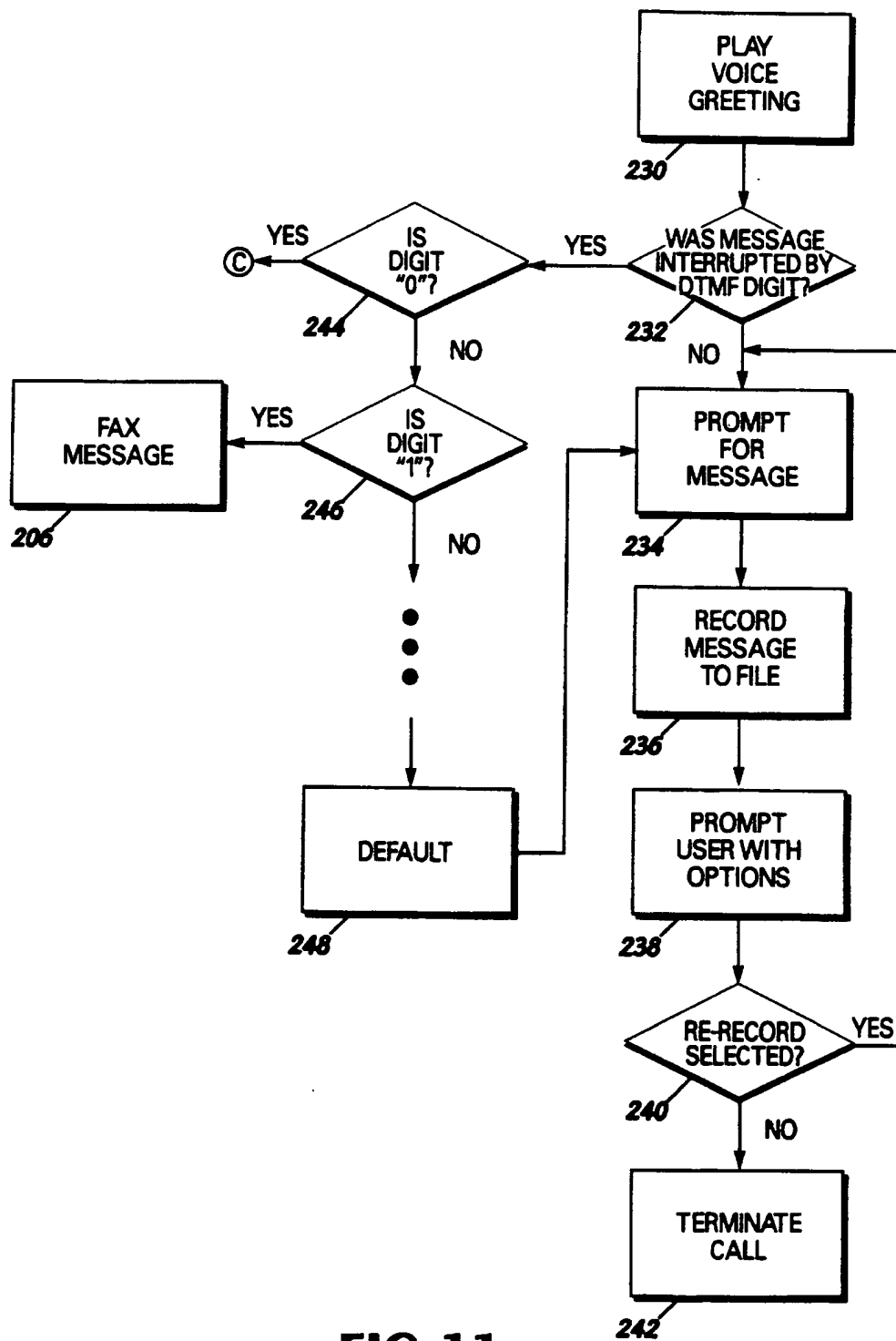


FIG 11

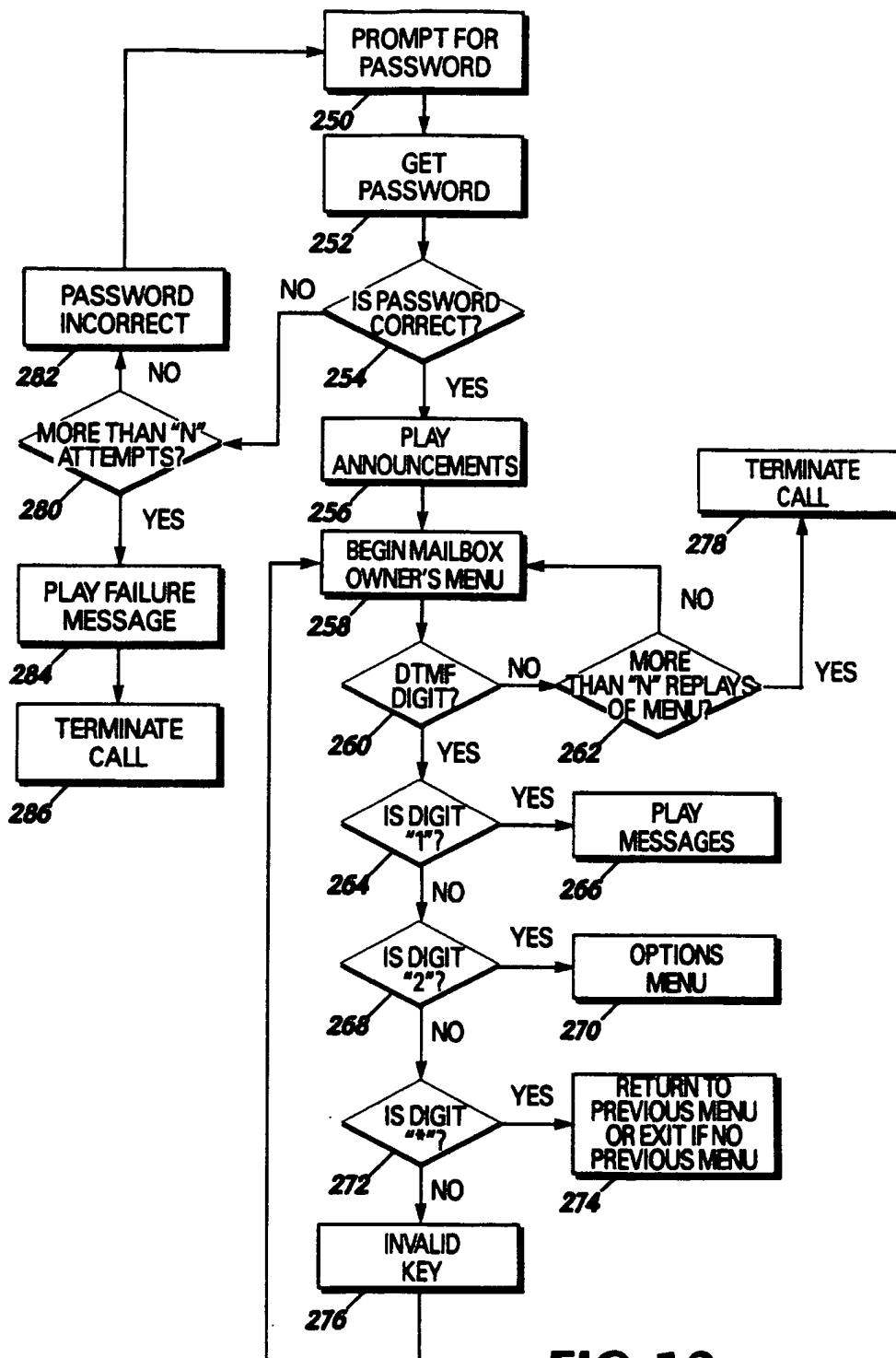
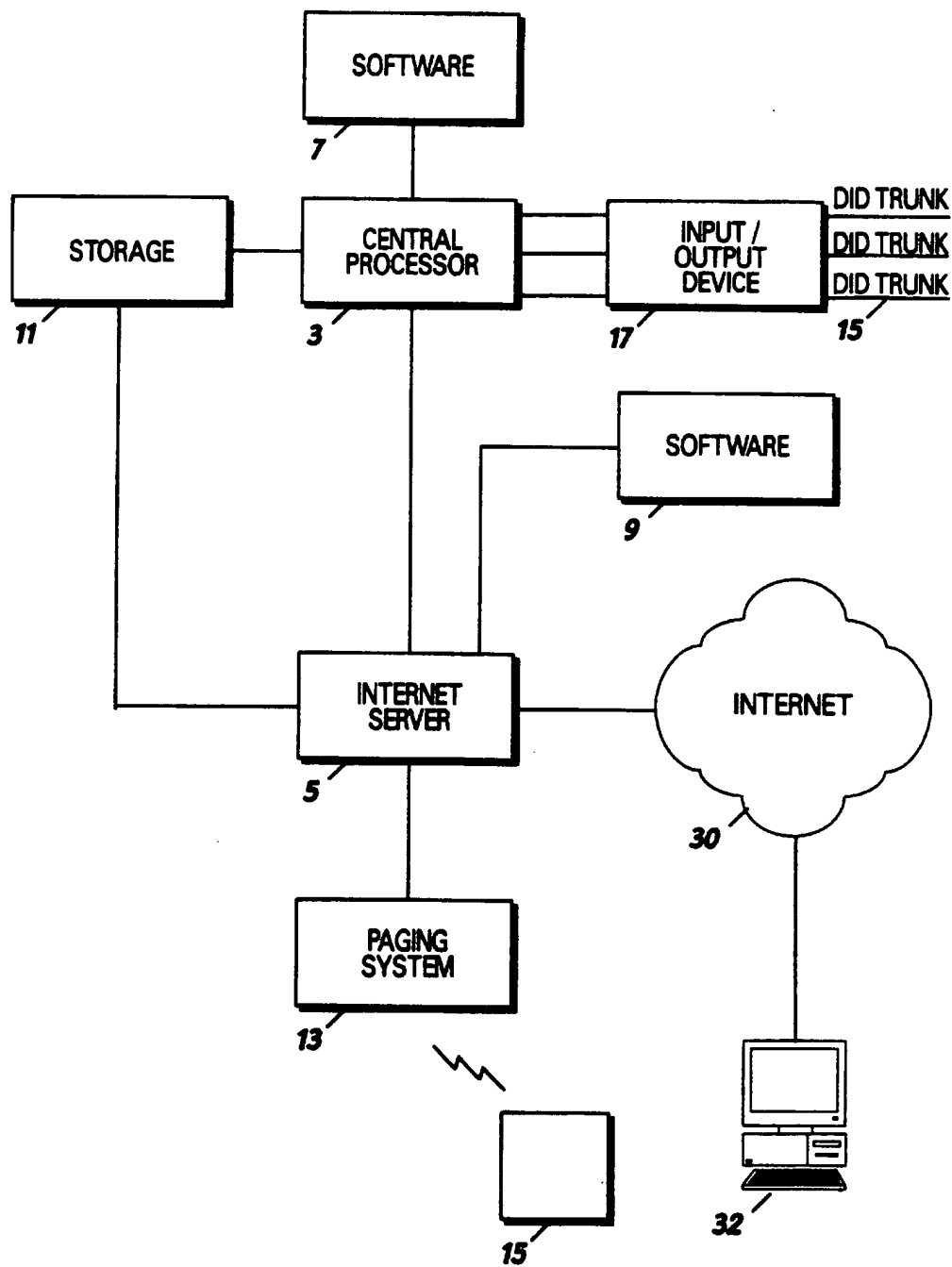
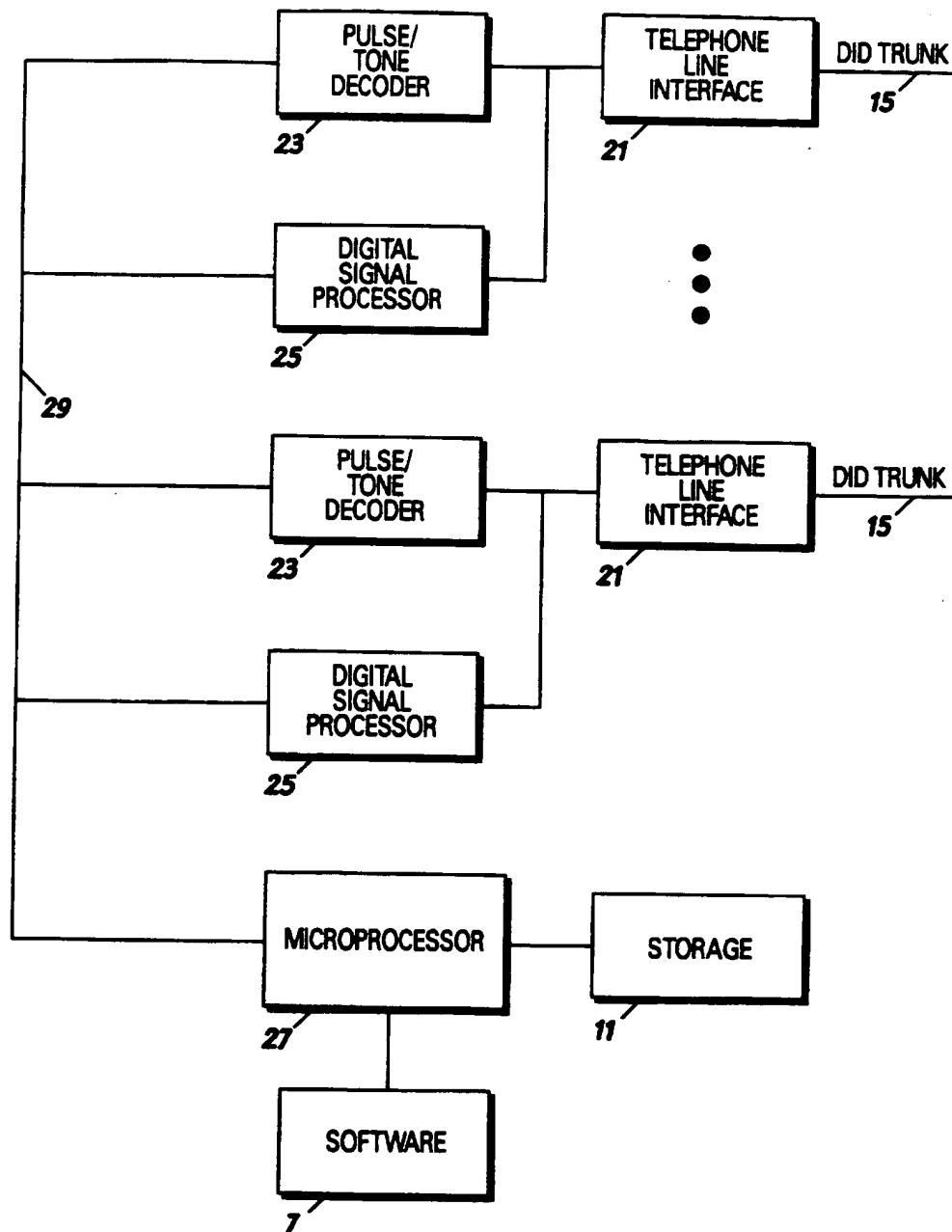
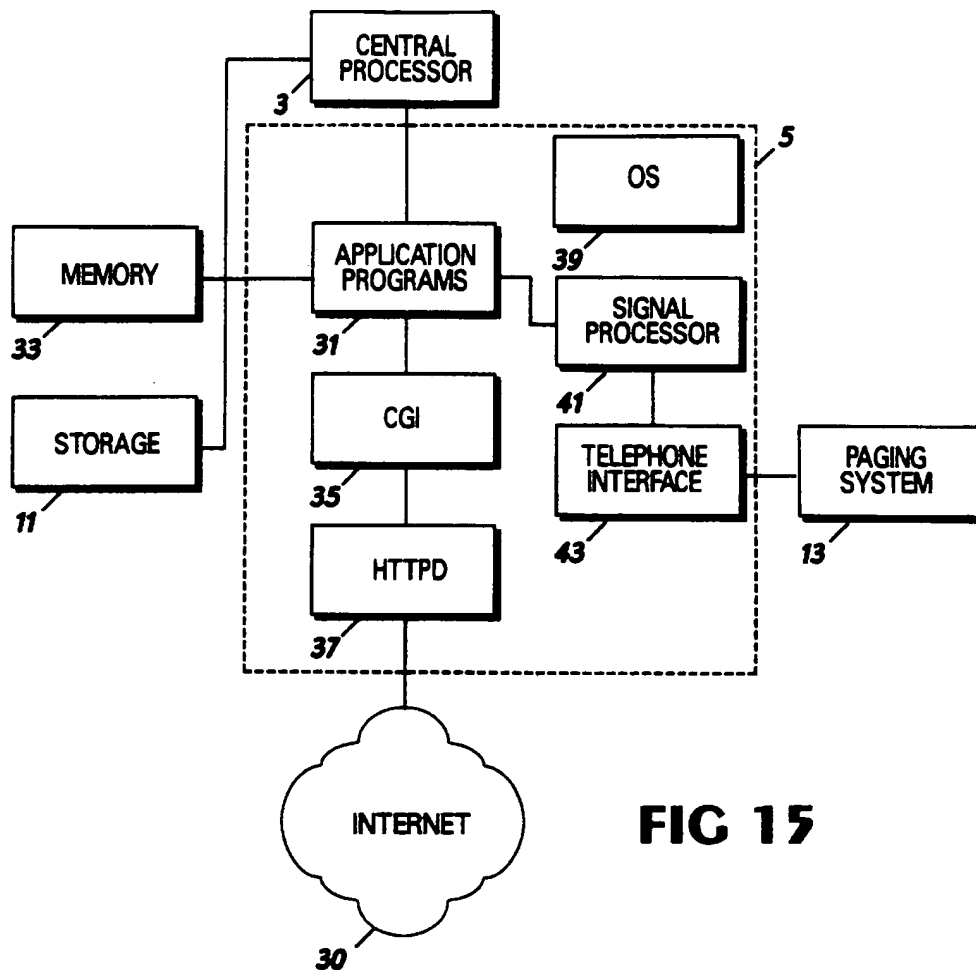


FIG 12

**FIG 13**

**FIG 14**



INDIVIDUAL APPLICATION PROGRAMS
COMMON GATEWAY INTERFACE (CGI)
HTTPD
INTERNET DEAMON (INETD)
OPERATING SYSTEM (OS)
TCP/IP

FIG 16A

PREFORMATTED HTML FILE
HTTPD
INETD
OS
TCP/IP

FIG 16B

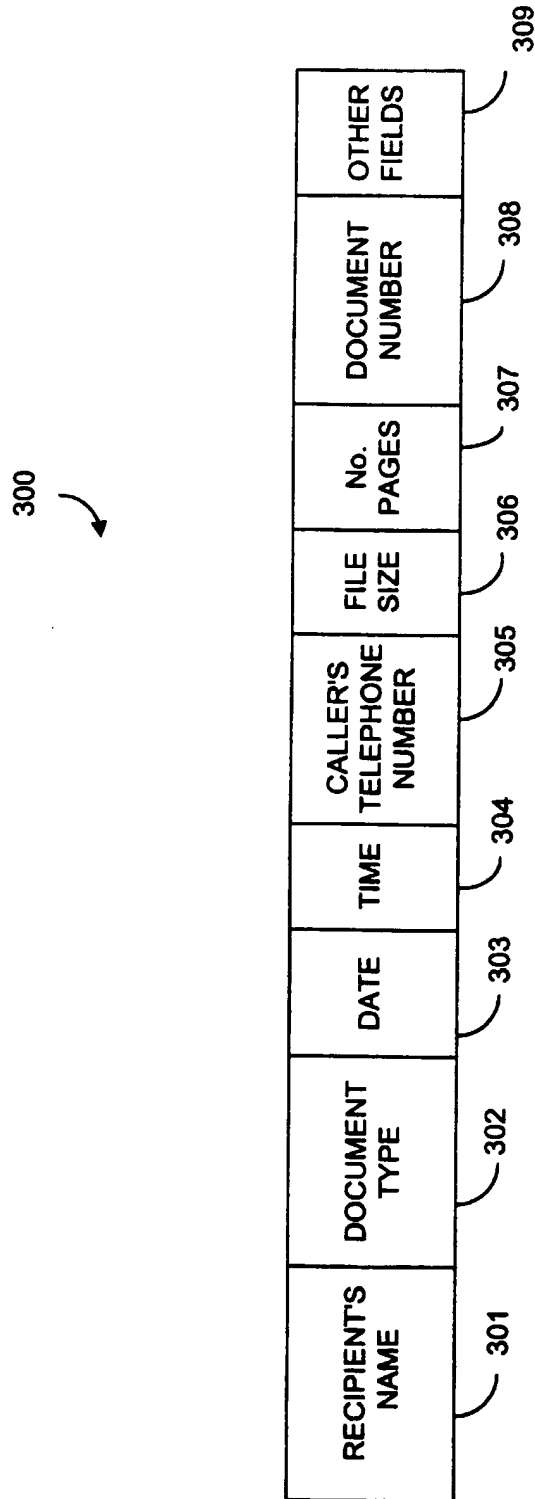


FIG. 17

320

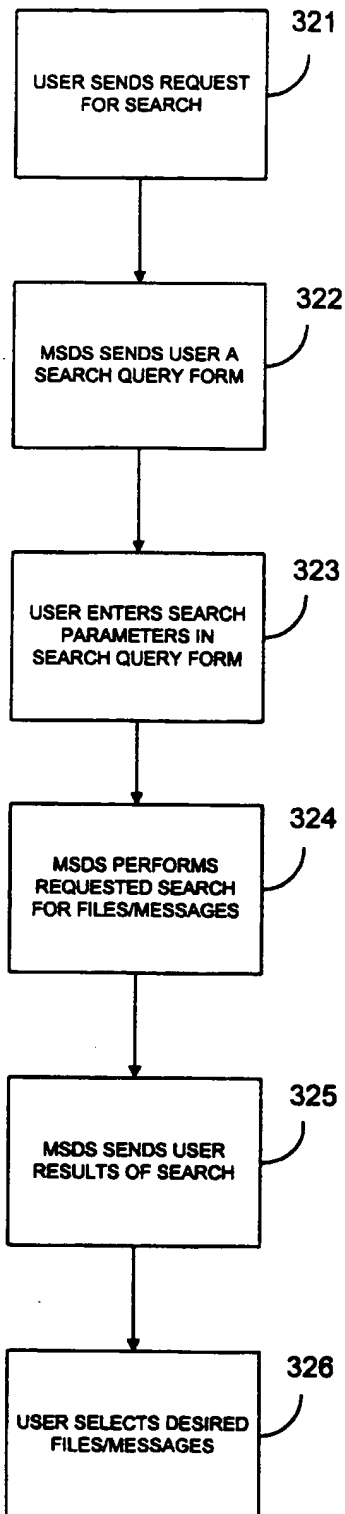


FIG. 18




SEARCH QUERY	
RECIPIENT'S NAME:	<input type="text"/> 
DOCUMENT TYPE:	<input type="text"/> 
DATE:	<input type="text"/>
TIME:	<input type="text"/>
CALLING NO.:	<input type="text"/>
FILE SIZE:	<input type="text"/>
NO. PAGES:	<input type="text"/>
DOCUMENT NO.:	<input type="text"/>
OTHER FIELD:	<input type="text"/> 
<u>SEARCH</u>	<u>RECENT FILES</u>
<u>STORED</u> <u>SEARCH GROUP</u>	<u>HELP</u>

FIG. 19




SEARCH QUERY	
RECIPIENT'S NAME:	<input type="text"/> 
DOCUMENT TYPE:	<input type="text" value="FACSIMILE"/> 
DATE:	<input type="text"/>
TIME:	<input type="text"/>
CALLING NO.:	<input type="text" value="(404) 249-6801"/>
FILE SIZE:	<input type="text"/>
NO. PAGES:	<input type="text"/>
DOCUMENT NO.:	<input type="text"/>
OTHER FIELD:	<input type="text"/> 
<div><div><u>SEARCH</u></div><div><u>RECENT FILES</u></div></div> <div><div><u>STORED SEARCHES</u></div><div><u>HELP</u></div></div>	

FIG. 20

SEARCH RESULTS

1. Document No. 11: Facsimile from (404) 249-6801 to Jane Doe on May 31, 1995. 3 Pages
2. Document No. 243: Facsimile from (404) 249-6801 to Jane Doe on July 16, 1995. 21 Pages
3. Document No. 1002: Facsimile from (404) 249-6801 to Jane Doe on January 1, 1996. 10 Pages

SAVE SEARCH AS:

CHARLES R. BOBO FACSIMILES

HELP

FIG. 21

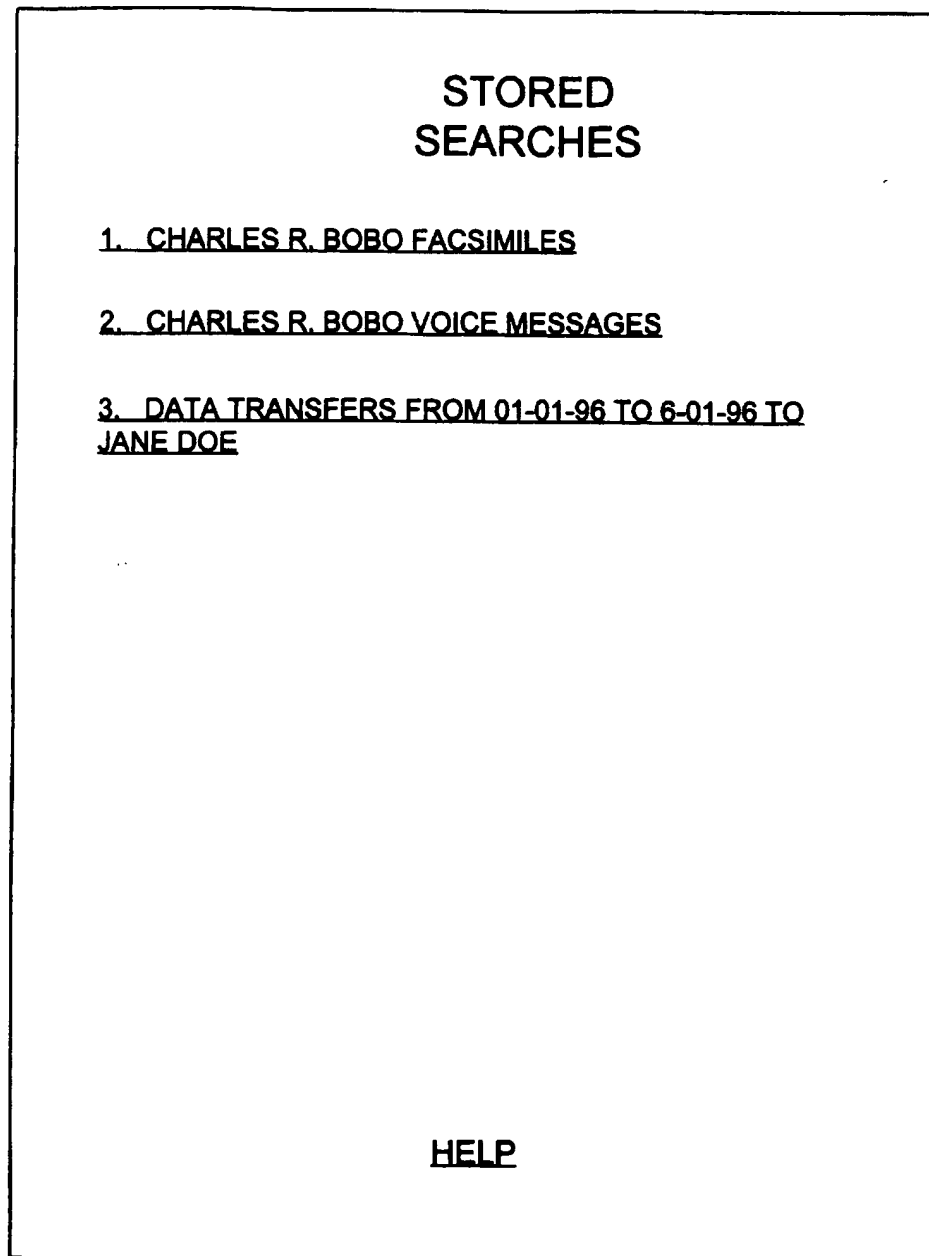


FIG. 22

SYSTEMS AND METHODS FOR STORING, DELIVERING, AND MANAGING MESSAGES

This application is a continuation-in-part of patent application Ser. No. 08/431,716, filed Apr. 28, 1995, now U.S. Pat. No. 5,675,507.

FIELD OF THE INVENTION

This invention relates to system(s) and method(s) for storing and delivering messages and, more particularly, to system(s) and method(s) for storing messages and for delivering the messages through a network, such as the Internet, or a telephone line to an intended recipient. In another aspect, the invention relates to system(s) and method(s) for storing, delivering, and managing messages or other files, such as for archival purposes or for document tracking.

BACKGROUND OF THE INVENTION

Even though the facsimile machine is heavily relied upon by businesses of all sizes and is quickly becoming a standard piece of office equipment, many businesses or households cannot receive the benefits of the facsimile machine. Unfortunately, for a small business or for a private household, a facsimile machine is a rather expensive piece of equipment. In addition to the cost of purchasing the facsimile machine, the facsimile machine also requires toner, paper, maintenance, as well as possible repairs. These expenses may be large enough to prevent many of the small businesses and certainly many households from benefiting from the service that the facsimile machine can provide. For others who are constantly traveling and who do not have an office, it may be impractical to own a facsimile machine. In fact, the Atlanta Business Chronicle estimates that 30% of the small businesses do not have any facsimile machines. Therefore, many businesses and households are at a disadvantage since they do not have access to a facsimile machine.

Because a facsimile machine can be such an asset to a company and is heavily relied upon to quickly transmit and receive documents, a problem exists in that the machines are not always available to receive a facsimile message. At times, a facsimile machine may be busy receiving another message or the machine may be transmitting a message of its own. During these times, a person must periodically attempt to send the message until communication is established with the desired facsimile machine. This inability to connect with a facsimile machine can be frustrating, can consume quite a bit of the person's time, and prevent the person from performing more productive tasks. While some more advanced facsimile machines will retry to establish communication a number of times, a person will still have to check on the facsimile machine to ensure that the message was transmitted or to re-initiate the transmission of the message.

In addition to labor costs and a reduction in office efficiency, a facsimile machine may present costs to businesses that are not readily calculated. These costs include the loss of business or the loss of goodwill that occurs when the facsimile machine is not accessible by another facsimile machine. These costs can occur for various reasons, such as when the facsimile machine is out of paper, when the machine needs repairing, or when the facsimile machine is busy with another message. These costs occur more frequently with some of the smaller businesses, who are also less able to incur these expenses, since many of them have a single phone line for a telephone handset and the facsimile machine and thereby stand to lose both telephone calls and

facsimile messages when the single line is busy. In fact, the Atlanta Business Chronicle estimated that fewer than 5% of the small businesses have 2 or more facsimile machines. Many of the larger companies can reduce these losses by having more than one facsimile machine and by having calls switched to another machine when one of the machines is busy. These losses, however, cannot be completely eliminated since the machines can still experience a demand which exceeds their capabilities.

A main benefit of the facsimile machine, namely the quick transfer of documents, does not necessarily mean that the documents will quickly be routed to the intended recipient. The facsimile machines may be unattended and a received facsimile message may not be noticed until a relatively long period of time has elapsed. Further, even for those machines which are under constant supervision, the routing procedures established in an office may delay the delivery of the documents. It is therefore a problem in many offices to quickly route the facsimile message to the intended recipient.

The nature of the facsimile message also renders it difficult for the intended recipient to receive a sensitive message without having the message exposed to others in the office who can intercept and read the message. If the intended recipient is unaware that the message is being sent, other people may see the message while it is being delivered or while the message remains next to the machine. When the intended recipient is given notice that a sensitive message is being transmitted, the intended recipient must wait near the facsimile machine until the message is received. It was therefore difficult to maintain the contents of a facsimile message confidential.

In an office with a large number of employees, it may also be difficult to simply determine where the facsimile message should be routed. In light of this difficulty, some systems have been developed to automatically route facsimile messages to their intended recipient. One type of system, such as the one disclosed in U.S. Pat. No. 5,257,112 to Okada, can route an incoming call to a particular facsimile machine based upon codes entered with telephone push-buttons by the sender of the message. Another type of system, such as the one disclosed in U.S. Pat. No. 5,115,326 to Burgess et al. or in U.S. Pat. No. 5,247,591 to Baran, requires the sender to use a specially formatted cover page which is read by the system. This type of system, however, burdens the sender, who may very well be a client or customer, by requiring the sender to take special steps or additional steps to transmit a facsimile message. These systems are therefore not very effective or desirable.

Another type of routing system links a facsimile machine to a Local Area Network (LAN) in an office. For instance, in the systems disclosed in the patents to Baran and Burgess et al., after the system reads the cover sheet to determine the intended recipient of the facsimile message, the systems send an E-mail message to the recipient through the local network connecting the facsimile machine to the recipient's computer. Other office systems, such as those in U.S. Pat. No. 5,091,790 to Silverberg and U.S. Pat. No. 5,291,546 to Giler et al., are linked to the office's voice mail system and may leave a message with the intended recipient that a facsimile message has been received. Some systems which are even more advanced, such as those in U.S. Pat. No. 5,317,628 to Misholi et al. and U.S. Pat. No. 5,333,266 to Boaz et al., are connected to an office's local network and provide integrated control of voice messages, E-mail messages, and facsimile messages.

The various systems for routing facsimile messages, and possibly messages of other types received in the office, are

3

very sophisticated and expensive systems. While these office systems are desirable in that they can effectively route the messages at the office to their intended recipients, the systems are extremely expensive and only those companies with a great number of employees can offset the costs of the system with the benefits that the system will provide to their company. Thus, for most businesses, it still remains a problem to effectively and quickly route messages to the intended recipients. It also remains a problem for most businesses to route the messages in a manner which can preserve the confidential nature of the messages.

Even for the businesses that have a message routing system and especially for those that do not have any type of system, it is usually difficult for a person to retrieve facsimile messages while away from the office. Typically, a person away on business must call into the office and be informed by someone in the office as to the facsimile messages that have been received. Consequently, the person must call into the office during normal business hours while someone is in the office and is therefore limited in the time that the information in a facsimile message can be relayed.

If the person away on business wants to look at the facsimile message, someone at the office must resend the message to a facsimile machine accessible to that person. Since this accessible machine is often a facsimile machine at another business or at a hotel where the person is lodging, it is difficult for the person to receive the facsimile message without risking disclosure of its contents. Further, since someone at the person's office must remember to send the message and since someone at the accessible facsimile machine must route the message to the person away from the office, the person may not receive all of the facsimile messages or may have to wait to receive the messages.

The retrieval of facsimile messages, as well as voice mail messages, while away from the office is not without certain costs. For one, the person often must incur long distance telephone charges when the person calls the office to check on the messages and to have someone in the office send the messages to another facsimile. The person will then incur the expenses of transmitting the message to a fax bureau or hotel desk as well as the receiving location's own charges for use of their equipment. While these charges are certainly not substantial, the charges are nonetheless expenses incurred while the person is away from the office.

Overall, while the facsimile machine is an indispensable piece of equipment for many businesses, the facsimile machine presents a number of problems or costs. Many businesses or households are disadvantaged since they are unable to reap the benefits of the facsimile machine. For the businesses that do have facsimile machines, the businesses must incur the normal costs of operating the facsimile machine in addition to the costs that may be incurred when the facsimile machine or machines are unable to receive a message. Further, the facsimile messages may not be efficiently or reliably routed to the intended recipient and may have its contents revealed during the routing process. The costs and problems in routing a facsimile message are compounded when the intended recipient is away from the office.

Many of the problems associated with facsimile messages are not unique to just facsimile messages but are also associated with voice mail messages and data messages. With regard to voice messages, many businesses do not have voice mail systems and must write the message down. Thus, the person away from the office must call in during normal office hours to discover who has called. The information in

4

these messages are usually limited to just the person who called, their number, and perhaps some indication as to the nature of the call. For those businesses that have voice mail, the person away from the office must call in and frequently incur long distance charges. Thus, there is a need for a system for storing and delivery voice messages which can be easily and inexpensively accessed at any time.

With regard to data messages, the transmission of the message often requires some coordination between the sender and the recipient. For instance, the recipient's computer must be turned on to receive the message, which usually occurs only when someone is present during normal office hours. Consequently, the recipient's computer is usually only able to receive a data message during normal office hours. Many households and also businesses may not have a dedicated data line and must switch the line between the phone, computer, and facsimile. In such a situation, the sender must call and inform the recipient to switch the line over to the computer and might have to wait until the sender can receive the message. The retransmission of the data message to another location, such as when someone is away from the office, only further complicates the delivery. It is therefore frequently difficult to transmit and receive data messages and is also difficult to later relay the messages to another location.

A standard business practice of many companies is to maintain records of all correspondence between itself and other entities. Traditionally, the correspondence that has been tracked and recorded includes letters or other such printed materials that is mailed to or from a company to the other entity. Although tracking correspondence of printed materials is relatively easy, non-traditional correspondence, such as facsimile messages, e-mail messages, voice messages, or data messages, are more difficult to track and record.

For example, facsimile messages may be difficult to track and record since the messages may be received on thermal paper, which suffers from a disadvantage that the printing fades over time. Also, accurate tracking of facsimile messages is difficult since the facsimile messages may only be partially printed at the facsimile machine or the messages may be lost or only partially delivered to their intended recipients. Facsimile messages also present difficulties since they are often delivered within an organization through different channels than ordinary mail and thus easily fall outside the normal record keeping procedures of the company.

Voice mail messages are also difficult to track and record. Although voice messages can be saved, many voice mail servers automatically delete the messages after a certain period of time. To maintain a permanent record of a voice message, the voice message may be transcribed and a printed copy of the message may be kept in the records. This transcribed copy of the voice message, however, is less credible and thus less desirable than the original voice message since the transcribed copy may have altered material or may omit certain portions of the message.

In addition to facsimile and voice mail messages, data messages are also difficult to track and record. A download or upload of a file may only be evident by the existence of a file itself. A file transfer procedure normally does not lend itself to any permanent record of what file was transferred, the dialed telephone number, the telephone number of the computer receiving the file, the time, or the date of the transfer. It is therefore difficult to maintain accurate records of all data transfers between itself and another entity.

SUMMARY OF THE INVENTION

It is an object of the invention to reliably and efficiently route messages to an intended recipient.

It is another object of the invention to route messages to the intended recipient while maintaining the contents of the message confidential.

It is another object of the invention to enable the intended recipient to access the messages easily and with minimal costs.

It is a further object of the invention to permit the simultaneous receipt of more than one message on behalf of the intended recipient.

It is a further object of the invention to enable the intended recipient of a message to access the message at any time and at virtually any location world-wide.

It is yet a further object of the invention to enable the intended recipient of a message to browse through the received messages.

It is yet a further object of the invention to quickly notify an intended recipient that a message has been received.

It is still another object of the invention to receive messages of various types.

It is still another object of the invention to deliver messages according to the preferences of the intended recipient.

It is still a further object of the invention to record and track correspondence, such as facsimile messages, voice mail messages, and data transfers.

Additional objects, advantages and novel features of the invention will be set forth in the description which follows, and will become apparent to those skilled in the art upon reading this description or practicing the invention. The objects and advantages of the invention may be realized and attained by the appended claims.

To achieve the foregoing and other objects, in accordance with the present invention, as embodied and broadly described herein, a system and method for storing and delivering messages involves receiving an incoming call and detecting an address signal associated with the incoming call, the address signal being associated with a user of the message storage and delivery system. A message accompanied with the address signal is then received and converted from a first file format to a second file format. The message is stored in the second file format within a storage area and is retrieved after a request has been received from the user. At least a portion of the message is then transmitted to the user over a network with the second file format being a mixed media page layout language.

In another aspect, a network message storage and delivery system comprises a central processor for receiving an incoming call, for detecting an address signal on the incoming call, for detecting a message on the incoming call, and for placing the message in a storage area. The address signal on the incoming call is associated with a user of the network message storage and delivery system. A network server receives the message from the storage area, converts the message into a mixed media page layout language, and places the message in the storage area. When the network server receives a request from the user over the network, the network server transmits at least a portion of the message over the network to the user.

Preferably, the network storage and delivery system can receive facsimile messages, data messages, or voice messages and the network is the Internet. The messages are converted into a standard generalized mark-up language and

the user is notified that a message has arrived through E-mail or through a paging system. A listing of the facsimile messages may be sent to the user in one of several formats. These formats include a textual only listing or a listing along with a full or reduced size image of the first page of each message. A full or reduced size image of each page of a message in the listing may alternatively be presented to the user.

According to a further aspect, the invention relates to a system and method for managing files or messages and involves storing message signals in storage and receiving requests from a user for a search. The search preferably comprises a search query that is completed by a user and supplied to a hyper-text transfer protocol daemon (HTTPD) in the system. The HTTPD transfers the request through a common gateway interface (CGI) to an application program which conducts the search. The results of the search are preferably returned through the HTTPD to the computer in the form of a listing of all messages or files satisfying the search parameters. The user may then select one or more of the listed messages or files and may save the search for later references.

BRIEF DESCRIPTION OF THE DRAWINGS

The accompanying drawings, which are incorporated in, and form a part of, the specification, illustrate an embodiment of the present invention and, together with the description, serve to explain the principles of the invention. In the drawings:

FIG. 1 is a block diagram illustrating the connections of a message storage and delivery system MSDS;

FIG. 2 is an overall flow chart of operations for transmitting a message to the MSDS of FIG. 1;

FIG. 3 is an overall flow chart of operations for receiving a message stored at the MSDS of FIG. 1;

FIGS. 4(A) and (B) are flowcharts of operations for generating HTML files according to user preferences;

FIG. 5 is a flowchart of operations for generating requested information;

FIG. 6 is a flowchart of operations for converting a facsimile message into HTML files;

FIG. 7 is an exemplary display of a first page of a facsimile message according to a fourth display option;

FIG. 8 is a flowchart of operations for converting a voice message into an HTML file;

FIG. 9 is a flowchart of operations for converting a data message into an HTML file;

FIG. 10 is a flowchart of operations for detecting a type of call received at the MSDS 10;

FIG. 11 is a flowchart of operations for receiving voice messages;

FIG. 12 is a flowchart of operations for interacting with an owner's call;

FIG. 13 is a more detailed block diagram of the MSDS 10;

FIG. 14 is a block diagram of the central processor in FIG. 13;

FIG. 15 is a block diagram of the Internet Server of FIG. 13;

FIGS. 16(A) and 16(B) depict possible software layers for the Internet Server of FIG. 13;

FIG. 17 is a diagram of a data entry for a message signal;

FIG. 18 is a flowchart of a process for sending a search query, for conducting a search, and for returning results of the search to a computer through the Internet;

7

FIG. 19 is an example of a search query form for defining a desired search;

FIG. 20 is an example of a completed search query;

FIG. 21 is an example of a set of search results returned to the computer in response to the search query of FIG. 20; and

FIG. 22 is an example of a listing of stored searches.

DETAILED DESCRIPTION OF THE PREFERRED EMBODIMENTS

Reference will now be made in detail to the preferred embodiments of the invention, examples of which are illustrated in the accompanying drawings.

With reference to FIG. 1, a message storage and delivery system (MSDS) 10 is connected to a central office 20 of the telephone company through at least one direct inward dialing (DID) trunk 15. With each call on the DID trunk 15, an address signal indicating the telephone number being called is provided to the MSDS 10. The DID trunk 15 can carry a large number of telephone numbers or addresses. Preferably, the DID trunk 15 comprises a number of DID trunks 15 connected in parallel between the central office 20 and the MSDS 10 so that the MSDS 10 can simultaneously receive more than one call and, moreover, can simultaneously receive more than one call for a single telephone number or address.

The central office 20 is connected to a number of third parties. For instance, the central office 20 may be connected to a facsimile machine 24, a telephone set 26, and to a computer 28 with each connection being made through a separate telephone line. While a single computer 28 is shown in the figure, the single computer 28 may actually represent a local area network which is connected through the central office 20 to the MSDS 10. Although the facsimile machine 24, telephone set 26, and computer 28 have been shown on separate lines, it should be understood that one or more of these devices could share a single line.

The MSDS 10 is also connected to a network, preferably the Internet World Wide Web 30. Although the Internet 30 has been shown as a single entity, it should be understood that the Internet 30 is actually a conglomeration of computer networks and is a constantly evolving and changing structure. The MSDS 10 therefore is not limited to the current structure or form of the Internet 30 but encompasses any future changes or additions to the Internet 30. Further, the MSDS 10 is shown as being directly connected to the Internet 30, such as through its own node or portal. The MSDS 10, however, may be practiced with any suitable connection to the Internet 30, such as through an intermediate Internet access provider.

With reference to FIG. 2 depicting an overall operation of the invention, a telephone call directed to a number serviced by the MSDS 10 is initiated at step 40 by a third party, for instance, through the facsimile machine 24, telephone set 26, or computer 28. The incoming telephone call may therefore carry a facsimile message, a voice message, or a data message. At step 42, the address signal associated with the initiated call is routed through the central office 20, over the DID trunk 15, and to the MSDS 10.

When the call reaches the MSDS 10, the call is routed within the MSDS 10 in a manner that will be described in more detail below with reference to FIG. 13. At step 46, the MSDS 10 answers the telephone call and receives the address signal from the DID trunk 15. Next, at step 48, the call is established between the MSDS 10 and the third party

8

and, at step 50, the MSDS 10 receives the message transmitted over the telephone line. The message is stored at step 52, a database within the MSDS 10 is updated at step 54, and the intended recipient of the message is notified at step 56. The intended recipient of the message uses the services provided by the MSDS 10 and will hereinafter be referred to as a user. At step 58, the message is converted into hyper-text mark-up language (HTML).

After the MSDS 10 receives a message for one of its users, the user can then communicate with the MSDS 10 at any time and at any location by connecting to the Internet World Wide Web 30 and retrieving the message stored within the MSDS 10. With reference to FIG. 3, at step 60 the user first connects to the Internet 30, such as through a personal computer 32 which may be connected to the Internet 30 in any suitable manner, such as through its own portal or node or through some intermediate access provider. The personal computer 32 is not limited to a single computer but may instead comprise a network of computers, such as a local area network within an office.

Once connected with the Internet 30, at step 62, the user accesses with a hyper-text browser the Universal Resource Locator (URL) associated with his or her MSDS 10 mailbox. The computer 32 may use any suitable hypertext browser, such as Netscape, to access the mailbox. A Hypertext Transfer Protocol Daemon (HTTPD) within the MSDS 10 receives the URL request at step 64 and, at step 66, requests user authentication. The user then supplies his or her ID and password at step 68 and, if found valid at step 70, the MSDS 10 provides the computer 32 with access to the mailbox at step 72. If the ID and password are invalid, as determined at step 70, then the HTTPD sends the computer 32 an authentication failure message at step 74.

After the user gains access to the mailbox at step 72, the user can request information stored within the MSDS 10. The MSDS 10 receives the request at step 76 and, at step 78, determines whether the information exists. As is common practice, the MSDS 10 also determines the validity of the request at step 78. The request from the user will include the mailbox number for the user, the message identifier, display preferences, and, if the message is a facsimile message, a page identifier. If for any reason the request is invalid, such as when a hacker is attempting to gain access to privileged information, the request for the information will be terminated.

If the requested information is available, then at step 80 the information is transmitted through the Internet 30 to the user's computer 32. If, on the other hand, the information does not exist, then at step 82 the MSDS 10 will generate the requested information and then send the information to the user's computer through the Internet 30 at step 80.

Prior to gaining access to the mailbox at step 72, the user is preferably sent a greeting page or other such type of information which permits the user to learn about the services provided by the MSDS 10, open an account with the MSDS 10, or gain access to an account. Once access is provided at step 72, the user is provided with information indicating the total number of messages stored in his or her mailbox within the MSDS 10. Preferably, the information sent by the MSDS 10 indicates the total number of messages for each type of message and also the total number of saved messages versus the total number of new messages.

The user is also preferably given the option at this step to change account information. The account information might include the E-mail address for the user, the manner in which messages are to be reviewed, the user's pager information,

as well as other user preferences. The display options and other user preferences will be discussed in further detail below.

The general information HTML file which indicates the total number of different messages is provided with a number of anchors, which are also termed links or references. In general, an anchor permits a user on the computer 32 to retrieve information located on another file. For instance, an anchor to a listing of facsimile messages is preferably provided on the display of the total number of messages. When the user selects the anchor for the facsimile list, the MSDS 10 pulls up and displays the file containing the list of facsimiles, such as a file "faxlist.html." The other types of messages, such as voice messages and data messages, would have similar anchors on the general information page directed to their respective HTML listing files.

When a new message is received at step 54 in FIG. 2, the user's mailbox is updated to display the total number and types of messages. The MSDS 10 might also update other files in addition to the total listing of messages. Additionally, at this time, the MSDS 10 sends an E-mail message to the user's computer 32 to inform the user of the newly arrived message. The MSDS 10 could also send notice to the user through a paging system so that the user receives almost instantaneous notice that a message is received.

The MSDS 10 also generates additional information according to the user's preferences. These preferences on how the MSDS 10 is configured for the user include options on how the messages are reviewed. With facsimile messages, for instance, the user can vary the amount or the type of information that will be supplied with the listing of the facsimile messages by selecting an appropriate option. Other options are also available so that the user can custom fit the MSDS 10 to the user's own computer 32 or own personal preferences.

For instance, when a facsimile message is received, the MSDS 10, at step 54, will update the total listing of all messages to indicate the newly received message and may additionally generate the HTML files for the newly received facsimile message according to the user's preferences. When the user later requests information on the message at step 76, the HTML information has already been generated and the MSDS 10 may directly send the requested information to the user at step 80. If, on the other hand, the user desires to view the message according to one of the other options, the MSDS 10 will generate the HTML files at step 82 according to that other option at the time of the request.

A first option available to the user for viewing a facsimile message is a textual only listing of the messages. The information on the textual listing preferably includes the date and time that the message was received at the MSDS 10, the telephone number from where the message was transmitted, the number of pages, the page size, and the size of the message in bytes. The messages, of course, could be listed with other types of information. When the user selects one of the facsimile messages on the list, a request is sent to the HTTPD within the MSDS 10 causing the message to be downloaded via the Internet 30 to the user's computer 32. Once the message is received by the computer 32, the message can be displayed, printed, or saved for further review.

The second through fifth options allow the user to preview an image of the facsimile message before having the message downloaded from the MSDS 10 through the Internet 30 and to the computer 32. The second option permits the user to view the list of messages with a reduced size image of the

cover page next to each entry on the list. When the user selects one of the messages on the list, the selected facsimile message is transmitted through the Internet 30 to the computer 32. The user may also scroll through the listings if all of the message cannot be displayed at one time on the computer 32.

The third option provides the user with a full size view of the cover page of each facsimile message. The user can quickly scroll through the cover pages of each message without downloading the entire message to the computer 32. The full size view of the cover pages permit the user to clearly discern any comments that may be placed on the cover page, which may not be possible from just a reduced image of the cover page available through the second option.

The fourth option provides the user with a reduced size image of each page and permits the user to scroll through the entire message. The user can therefore read the entire facsimile message on screen before the message is downloaded onto the computer 32. With this option, the user can go through the pages of the facsimile message and can also skip to the next message or previous message. Additionally, the user has the option of enlarging a page to a full size view of the page. When one of the messages is selected, as with the other options, the HTTPD within the MSDS 10 causes the facsimile message to be transmitted through the Internet 30 to the user's computer 32.

With a fifth option, a full size image of each page is transmitted to the user's computer 32. The user can scroll through the pages of the facsimile message and easily read the contents of each page. If the user wants the message downloaded to the computer 32, the user selects the message and the HTTPD within the MSDS 10 transmits the message to the user's computer 32 through the Internet 30.

As discussed above, after the database is updated at step 54, the MSDS 10 will generate additional information based upon the option selected for displaying the facsimile messages. More specifically, as shown in FIG. 4(A), if the first option has been selected, as determined at step 100, then at step 102 the MSDS 10 will generate the textual listing of the facsimile messages with anchors or references to the respective facsimile files. The HTML files are then moved to an Internet Server at step 104.

If the first option is not selected, the MSDS 10 next determines whether the second option has been selected at step 106. With the second option, the facsimile messages are listed along with a reduced size image of the cover page. To generate this information, the cover page is extracted from the facsimile file at step 108 and a reduced size HTML image of the cover page is created at step 110. At step 112, a listing of the facsimile messages is generated with a thumbnail view of each cover page linked to its respective facsimile file. The generated HTML files are then sent to the Internet Server at step 104.

When the third option is selected, as determined at step 114, a full size image of the cover page is sent to the computer 32. The full size image of the cover page is generated by first extracting the cover page from the facsimile file at step 116. Next, the cover page is converted into a full size HTML image at step 118 and, at step 120, the listing is generated with the embedded cover page linked to the facsimile file.

If, at step 122, the fourth option is determined to be selected, then a reduced size image of each page is provided to the user with the option of enlarging the page to view the contents of the page more clearly. With reference to FIG. 4(B), the information necessary for the third option is

11

produced by first extracting the first page of the facsimile message at step 124. A reduced size HTML image is created at step 126 and then a full size HTML image is created at step 128. At step 130, the listing is generated with embedded thumbnail images of the pages with links to the full size images. If the page is not the last page, as determined at step 140, then the next page is extracted at step 142 and steps 126 to 130 are repeated to generate the HTML files for the other pages of the facsimile message. After the last page has been converted into an HTML file according to the third option, the files are moved onto the Internet Server at step 104.

At step 144, the MSDS 10 determines whether the fifth option has been selected. The fifth option provides the user with a full size image of each page of the facsimile message. While only five options have been discussed, the invention may be practiced with additional options. Consequently, with additional options and with the fourth option not being selected, the MSDS 10 would next determine whether one of the additional options have been selected. With the preferred embodiment of the invention having only five options, however, the MSDS 10 will assume that the fifth option has been selected if none of the first four options were found to be selected.

The information necessary to display the pages of the facsimile message according to the fifth option is generated by first extracting the first page of the facsimile message at step 146. At step 148, a full size HTML image of the page is created and, at step 150, a listing is generated with an embedded image and links to previous and next pages. When the page is not the last page, as determined at step 152, the MSDS 10 extracts the next page and generates the HTML file for that page. After all pages have been converted into HTML files according to the fourth option, the files are sent to the Internet Server at step 104.

While FIGS. 4(A) and (B) describe the operations of the MSDS 10 at the time a message is received, FIG. 5 depicts an overall flowchart of operations for the MSDS 10 when the user requests a page of information in a display format other than the user's preferred option of displaying the message. FIG. 5 is therefore a more detailed explanation of how the MSDS 10 generates the necessary information at step 82 of FIG. 3.

In general, as shown in FIG. 5, the MSDS 10 first determines the type of image that is needed at step 82a. For example, at this step, the MSDS 10 will determine whether images are unnecessary, whether an image of just the cover page is necessary, whether an image is needed for every page, and whether the image needs to be a full size, a reduced size, or both full and reduced sized images. At step 82b, the MSDS 10 determines whether the image has already been created. If the image has not been created, then at step 82c the MSDS 10 will extract the page from the base facsimile file and, at step 82d, generate the required HTML image. As discussed above, the required image may be for just the cover page, for all the pages, and may be a full size and/or a reduced size image of the page. At step 82e, the image is embedded with links or anchors to other HTML files. These links or anchors might be references to the next and previous pages and also to the next and previous facsimile messages. Finally, the HTML file having the embedded image and links is sent to the user at step 80 in FIG. 3.

The process for converting a facsimile message into HTML files according to the fifth option will be described with reference to FIG. 6. This process will occur at step 54 when the message is received and when the fifth option is the

12

user's preferred option of displaying the messages. It should be understood that a similar type of process will also occur when the user requests a page of information according to the fifth option when the user is retrieving a facsimile message and the fifth option is not the user's preferred option. The conversion processes according to the other options will become apparent to those skilled in the art and will therefore not be discussed in further detail.

With reference to FIG. 6, when the facsimile message is received, the message is in a Tagged Image File Format/Facsimile (TIFF/F) and each page of the facsimile message is split into a separate file. Each page of the facsimile message is then converted from the TIFF/F format into a Portable Pixel Map (PPM) format. The PPM files are next converted into separate Graphic Interchange Format (GIF) files and then into separate HTML files. Thus, each page of the facsimile message is converted into a separate HTML file. The TIFF/F files may be converted into PPM with an available software package entitled "LIBTIFF" and the PPM files may be converted into GIF files with an available software package found in "Portable Pixel Map Tools."

The invention is not limited to this exact conversion process or to the particular software packages used in the conversion process. For instance, the TIFF/F files may be converted into another portable file format, through any other type of intermediate format, or may be converted directly into the GIF format. Further, instead of GIF, the facsimile messages may be converted into JPEG, BMP, PCX, PIF, PNG, or any other suitable type of file format.

The files may be identified with any suitable filename. In the preferred embodiment, the files for each user are stored in a separate directory assigned to just that one user because an entire directory for a given user generally can be protected easier than the individual files. The memory, however, may be organized in other ways with the files for a single user being stored in different directories. The first part of the filename is a number preferably sequentially determined according to the order in which messages arrive for that user. The preferred naming convention for ending the filenames is depicted in FIG. 6. Each page of the facsimile message is saved as a separate file with an extension defined by the format of the file. Thus, the files will end with an extension of ".TIFF," ".PPM," ".GIF" or ".HTML" according to the format of the particular file. In the example shown, the separate pages have filenames which end with the respective page number, for instance, the first page ends with a "1." The files, however, are preferably terminated with a letter or multiples letters to indicate the order of the pages. For instance, page 1 might have an ending of "aa," page 2 might have an ending of "ab," etc. The invention, however, is not limited to the disclosed naming convention but encompasses other conventions that will be apparent to those skilled in the art.

As shown in FIG. 6, in addition to the GIF files representing the pages of the facsimile message, the HTML files include a number of anchors or references. In the example shown, the first HTML file has an anchor a for the "Next Page." Anchor a is defined as `a=Next Page` and will therefore reference the second HTML file when a user selects the "Next Page." The second HTML file has an anchor b for the "Previous Page" and an anchor c for the "Next Page" and the third HTML file has an anchor d for the "Previous Page." With these particular HTML files, the user can scroll through each page of the facsimile message and view a full size image of the page.

Each HTML file preferably contains anchors in addition to those relating to "Next Page" and "Previous Page." For

13

instance, each HTML file may contain an anchor to the next facsimile message, an anchor to the previous facsimile message, and an anchor to return to the facsimile list. The HTML files preferably contain anchors relating to "Save" and "Delete." When the "Save" anchor is selected, the user would be able to save the message under a more descriptive name for the message. The "Delete" anchor is preferably followed by an inquiry as to whether the user is certain that he or she wants to delete the message. Other anchors, such as an anchor to the general listing, will be apparent to those skilled in the art and may also be provided.

FIG. 7 provides an example of a display according to the fifth option for the first page of the facsimile message shown in FIG. 6. The headings of the display provide information on the telephone number from where the message was sent, the date and time the message was received at the MSDS 10, and an indication of the page of the message being displayed. The main portion of the display is the full size image of the page. At the bottom of the display, an anchor or link is provided to the "Next Page" and another anchor is provided to the "Return to Fax Listing." Additional information may also be provided on the display, such as a link to a company operating the MSDS 10.

An example of the "1.html" file for generating the display shown in FIG. 7 is shown below in Table 1.

```
<HTML>
<HEAD>
<TITLE>Fax Received on May 31, 1995 at 1:58 PM from
(404)249 6801; Page 1 of 3</TITLE>
</HEAD>
<BODY>
<H1>Fax from (404)249-6801</H1>
<H2>Received on May 31, 1995 at 1:58 PM</H2>
<H2>Page 1 of 3</H2>
<IMG SRC="1.gif">
<P>
<A HREF="2.html">Next Page</a>
<HR>
<A HREF="faxlist.html">Return to Fax Listing</A>
<P>
This page was automatically generated by FaxWeb(tm) on
May 31, 1995 at 2:05 PM.
<P>
&copy; 1995 NetOffice, Inc.
<HR>
<Address>
<A HREF="http://www.netoffice.com/">NetOffice,
Inc.</A><BR>
PO Box 7115<BR>
Atlanta, Ga. 30357<BR>
<A
HREF=
"mailto:info@netoffice.com">info@netoffice.com</A>
</Address>
</BODY>
</HTML>
```

TABLE 1

As is apparent from the listing in Table 1, the image file "1.gif" for the first page is embedded into the HTML file "1.html." Also apparent from the listing is that the anchor for "Next Page" directs the MSDS 10 to the second page of the facsimile message having the filename "2.html" and the

14

anchor for "Return to Fax Listing" directs the MSDS 10 to the filename "faxlist.html" containing the list of facsimile messages.

A process for converting a voice message into an HTML file is illustrated in FIG. 8. The voice message is originally stored in a VOX format or an AD/PCM format and is retrieved at step 170. The voice message is then converted either into an AU format or WAV format in accordance with the user's preference, which is stored in memory. Preferably, the message is preferably in the AD/PCM format originally and is converted in WAV, but the voice files may alternatively be stored and converted in file formats other than the ones disclosed, such as RealAudio (RA).

At step 174, the listing of all of the voice messages is then updated to include a listing of the newly received voice message and an anchor to the voice message. For instance, the original voice message may be stored with filename "1.vox" and is converted into WAV and stored with a filename "1.wav." The HTML file "voicelist.html" which contains a list of all voice messages would then have an anchor to the filename "1.wav" along with identifying information for the voice message, such as when the message was received.

The listing of the voice messages may have additional anchors or references. For instance, each voice message may have an anchor directing the MSDS 10 to a file which contains a short sampling of the message. Thus, when the user selects this anchor, the user could receive the first 5 seconds of the message or some other predefined number of seconds. As with the listing of facsimile messages, the listing of the voice messages also preferably has anchors to "Save" and "Delete."

FIG. 9 illustrates a process for converting a data message into HTML. At step 180, the data file is retrieved from a database and at step 182 the HTML file containing the list of data messages is updated to include a listing of the newly received message along with identifying information. For instance, the HTML file for the listing "datalist.html" would be updated to include an anchor to a data file "file 1.1" and would have information such as the time and date that the data was transmitted, the size of the data file, as well as additional identifying information.

Because the MSDS 10 can receive messages of various types, such as a facsimile message, voice message or data message, the MSDS 10 must be able to determine the type of message that is being sent over the DID trunk 15. With reference to FIG. 10, when an incoming call is received, the MSDS 10 goes off hook at step 200 and starts to generate a ringing sound. If, at step 202, a facsimile calling tone is detected, then the ringing sound is stopped at step 204 and the message is received as a facsimile message at step 206. Similarly, when a data modem calling tone is detected at step 208, the ringing sound is stopped at step 210 and the message is identified as a data message at step 212.

If the MSDS 10 detects a DTMF digit at step 214, the ringing sound is stopped at step 216 and the MSDS 10 then determines which digit was pressed. When the digit is a "1," as determined at step 218, the message is identified as a facsimile message. The MSDS 10 will thereafter receive and store the facsimile message in the manner described above with reference to FIG. 2. If the digit is identified as a "0" at step 220, the call is identified as an owner's call and will be processed in a manner that will be described below with reference to FIG. 12. As will be apparent, other digits may cause the MSDS 10 to take additional steps. If any other DTMF digit is pressed, at step 224 the MSDS 10 activates

15

a voice call system, which will be described in more detail below with reference to FIG. 11.

With step 226, the MSDS 10 will enter a loop continuously checking for a facsimile calling tone, a data modem calling tone, or for a DTMF digit. If after n rings none of these tones or digits has been detected, the ringing sound is stopped at step 228 and the voice call system is activated at step 224.

With reference to FIG. 11, when a fax calling tone or modem calling tone is not detected, the voice call system begins at step 230 by playing a voice greeting. If the greeting was not interrupted by a DTMF digit as determined at step 232, then the caller is prompted for the voice message at step 234 and, at step 236, the voice message is recorded and stored in memory. At step 238, the caller is prompted with a number of options, such as listening to the message, saving the message, or re-recording the message. Since the selection of these options with DTMF digits will be apparent to those skilled in the art, the details of this subroutine or subroutines will not be described in further detail. When the caller wishes to re-record the message, as determined at step 240, the caller is again prompted for a message at step 234. If the caller does not wish to re-record the message, the call is terminated at step 242.

If the voice greeting is interrupted by a DTMF digit, as determined at step 232, then the MSDS 10 ascertains which digit has been pressed. At step 244, if the digit is a "0," the MSDS 10 detects that the call is an owner's call. When the digit is a "1," the MSDS 10 is informed at step 206 that the call carries a facsimile message. As discussed above with reference to FIG. 10, other DTMF digits may cause the MSDS 10 to take additional steps. If an invalid digit is pressed, by default at step 248 the routine returns to step 234 of prompting the caller for a message.

It should be understood that the invention is not limited to the specific interactive voice response system described with reference to FIG. 11. As discussed above, the invention may be responsive to DTMF digits other than just a "0" and a "1." Further variations or alterations will be apparent to those skilled in the art.

With reference to FIG. 12, when the call is considered an owner's call, the caller is first prompted for the password at step 250. The password is received at step 252 and, if found correct at step 254, a set of announcements are played to the owner. These announcements would preferably inform the owner of the number of new messages that have been received, the number of saved messages, the number of facsimile message, the number of data messages, and the number of voice messages. Other announcements, of course, could also be made at this time.

At step 258, the owner then receives a recording of the owner's menu with the appropriate DTMF digit for each option. For instance, the DTMF digit "1" may be associated with playing a message, the DTMF digit "2" may be associated with an options menu, and the DTMF digit "*" may be associated with returning to a previous menu or terminating the call if no previous menu exists.

A DTMF digit is detected at step 260 and the appropriate action is taken based upon the digit received. Thus, if the digit is determined to be a "1" at step 264, the owner can play a message at step 266. At step 266, the owner is preferably greeted with a menu giving the owner the options of playing or downloading new messages, saved messages, facsimile messages, data messages, or voice messages. As should be apparent to those skilled in the art, the owner may receive one or more menus at step 266 and the owner may

16

enter one or more DTMF digits in order to play or download a particular message.

If, instead, the digit is determined to be a "2" at step 268, then the owner receives an options menu at step 270. With the options menu, the owner can enter or change certain parameters of the MSDS 10. For instance, the owner can change his or her password, the owner can change the manner in which facsimile messages are displayed on the computer 32, the owner can change the image file format from GIF to another format, the owner can select the file formats for the voice messages, as well as other options.

If the "*" DTMF digit is received, as determined at step 272, then the owner is returned to a previous menu. The "*" digit is also used to terminate the call when the owner has returned to the initial menu. The "*" digit is therefore universally recognized by the MSDS 10 throughout the various menus as a command for returning to a previous menu.

If the owner enters a DTMF digit that is not being used by the MSDS 10, the owner receives an indication at step 276 that the key is invalid and the owner is then again provided with the owner's menu at step 258. When the owner does not enter a DTMF digit while the owner's menu is being played, as determined at step 260, the menu will be replayed n times. Once the menu has been replayed n times, as determined at step 262, then the call will be terminated at step 278.

If the password is incorrect, as determined at step 254, then the MSDS 10 checks whether the user has made more than "n" attempts at step 280. If "n" attempts have not been made, then a password incorrect message will be displayed to the user at step 282 and the user will once again be prompted for the password at step 250. When the user has made "n" attempts to enter the correct password, the MSDS 10 will play a failure message to the user at step 284 and then terminate the call at step 286. The specific number "n" may be three so that the call is terminated after three failed attempts.

The owner's menu may be responsive to an additional number of DTMF digits and may be structured in other ways. For instance, separate DTMF digits may direct the owner to the respective types of messages, such as a facsimile message, data message, or voice message. Also, separate DTMF digits may direct the owner to a recording of new messages or to a recording of saved messages. Other variations will be apparent to those skilled in the art.

A more detailed diagram of the MSDS 10 is shown in FIG. 13. As shown in the figure, a plurality of DID trunks 15 are received by an input/output device 17 and are then sent to a central processor 3. The number of DID trunks 15 may be changed to any suitable number that would be necessary to accommodate the anticipated number of telephone calls that may be made to the MSDS 10. The input/output device 17 routes a call on one of the DID trunks 15 to an open port of the central processor 3 and is preferably a DID Interface Box manufactured by Exacom.

The central processor 3 receives the calls on the DID trunks 15 and stores the messages in storage 11 in accordance with software 7. Preferably, a separate directory in storage 11 is established for each user having an account on the MSDS 10 so that all of the messages for a single user will be stored in the same directory. It should be understood that the number of processors within the central processor 3 is dependent upon the number of DID trunks 15. With a greater number of DID trunks 15 capable of handling a larger number of telephone calls, the central processor 3 may actually comprise a number of computers. The input/output

17

device 17 would then function to route incoming calls to an available computer within the central processor 3.

A more detailed diagram of the central processor 3 is shown in FIG. 14. The central processor 3 comprises a telephone line interface 21 for each DID trunk 15. The telephone interface 21 provides the ringing sounds and other communication interfacing with the telephone lines. The signals from the telephone interface 21 are routed to a pulse/tone decoder 23 and to a digital signal processor (DSP) 25. The pulse/tone decoder 23 detects the address signal off of an incoming call and sends the address signal onto a bus 29 to a microprocessor 27. The DSP performs the necessary signal processing on the incoming calls and routes the processed signals to the microprocessor 27.

The microprocessor 27 will then read the address signal from the pulse/tone decoder 23 and store the message from the DSP 25 in an appropriate directory in storage 11. As discussed above, the central processor 3 may comprise a number of computers or, more precisely, a number of microprocessors 27 with each microprocessor 27 handling the calls from a certain number, such as four, DID trunks 15. The microprocessor 27 may comprise any suitable microprocessor, but is preferably at least a 486 PC.

In addition to handling incoming calls and storing the messages in storage 11, the central processor 3 also coordinates the interactive voice response system of the MSDS 10. The software 7 would incorporate the flowcharts of operations for receiving a message shown in FIG. 3, for detecting the type of message on an incoming call shown in FIG. 10, for receiving voice messages shown in FIG. 11, and for receiving an owner's call shown in FIG. 12. Based upon the above-referenced flowcharts and the respective descriptions, the production of the software 7 is within the capability of one of ordinary skill in the art and will not be described in any further detail.

The Internet Server 5 is connected to the central processor 3, such as through a local area network, and also has access to the storage 11. The Internet Server 5 performs a number of functions according to software 9. For instance, the Internet Server 5 retrieves the data files stored in storage 11 by the central computer 3 and converts the files into the appropriate HTML files. The converted HTML files are then stored in storage 11 and may be downloaded to the computer 32 through the Internet 30. The Internet Server 5 also handles the requests from the computer 32, which might require the retrieval of files from the storage 11 and possibly the generation of additional HTML files.

The software 9 for the Internet Server 5 would therefore incorporate the flowchart of operations for generating HTML files according to user preferences shown in FIG. 4, for generating requested information from a user shown in FIG. 5, for converting facsimile messages into HTML shown in FIG. 6, for converting voice messages into HTML shown in FIG. 8, and for converting data messages into HTML shown in FIG. 9. Based upon the above-referenced flowcharts and their respective descriptions, the production of the software 9 is within the capability of one of ordinary skill in the art and need not be described in any further detail.

Nonetheless, a more detailed block diagram of the Internet Server 5 is shown in FIG. 15. The Internet Server 5 runs on a suitable operating system (OS) 39, which is preferably Windows NT. The Internet Server 5 has a number of application programs 31, such as the ones depicted in the flowcharts discussed above, for communicating with the central processor 3 and for accessing data from storage 11 and also from memory 33.

18

The memory 33, inter alia, would contain the data indicating the preferences of each user. Thus, for example, when a facsimile message in the TIFF/F format is retrieved by the Internet Server 5, the Internet Server 5 would ascertain from the data in memory 33 the preferred option of displaying the facsimile message and would generate the appropriate HTML files.

All interfacing with the Internet 30 is handled by the HTTPD 37, which, in the preferred embodiment, is "Enterprise Server" from NetScape Communications Corp. Any requests from users, such as a request for a file, would be handled by the HTTPD 37, transferred through the CGI 35, and then received by the application programs 31. The application programs 31 would then take appropriate actions according to the request, such as transferring the requested file through the CGI 35 to the HTTPD 37 and then through the Internet 30 to the user's computer 32.

The Internet Server 5 may be connected to a paging system 13. Upon the arrival of a new message, in addition to sending an E-mail message to the user's mailbox, the Internet Server 13 may also activate the paging system 13 so that a pager 15 would be activated. In this manner, the user could receive almost instantaneous notification that a message has arrived.

The paging system 13 is preferably one that transmits alphanumeric characters so that a message may be relayed to the user's pager 15. The Internet Server 5 therefore comprises a signal processor 41 for generating signals recognized by the paging system 13 and a telephone interface 43. The signal processor 41 preferably receives information from the application programs 31 and generates a paging message in a paging file format, such as XIO/TAP. The telephone interface 43 would include a modem, an automatic dialer, and other suitable components for communicating with the paging system 13.

The information from the application programs 31 may simply notify the user of a message or may provide more detailed information. For instance, with a facsimile message, the information from the application programs 31 may comprise CSI information identifying the sender's telephone number. The user would therefore receive a message on the pager 15 informing the user that a facsimile message was received from a specified telephone number. The amount and type of information that may be sent to the user on the pager 15 may vary according to the capabilities of the paging system 13 and may provide a greater or lesser amount of information than the examples provided.

The Internet Server 5 is not limited to the structure shown in FIG. 15 but may comprise additional components. For instance, the HTTPD 37 would be linked to the Internet 30 through some type of interface, such as a modem or router. The Internet Server 5 may be connected to the Internet 30 through typical phone lines, ISDN lines, a T1 circuit, a T3 circuit, or in other ways with other technologies as will be apparent to those skilled in the art.

Furthermore, the Internet Server 5 need not be connected to the Internet 30 but may be connected to other types of networks. For instance, the Internet Server 5, or more generally the network Server 5, could be connected to a large private network, such as one established for a large corporation. The network Server 5 would operate in the same manner by converting messages into HTML files, receiving requests for information from users on the network, and by transmitting the information to the users.

Also, at least one interface circuit would be located between the Internet Server 5 and the central processor 3 in

order to provide communication capabilities between the Internet Server 5 and the central processor 3. This network interface may be provided within both the Internet Server 5 and the central processor 3 or within only one of the Internet Server 5 or central processor 3.

Examples of the Internet Server 5 software layers are shown in FIGS. 16(A) and 16(B), with FIG. 16(A) representing the Internet Server 5 in an asynchronous mode of communication and FIG. 16(B) representing the Internet 5 in a synchronous mode of communication. As shown in the figures, the software 9 for the Internet Server 5 may additionally comprise an Internet Daemon for running the HTTPD 37. The software 9 for the Internet Server 5 would also include TCP/IP or other transport layers. Moreover, while the authentication is provided through the HTTPD 37, the authentication of the user's password and ID may be supplemented or replaced with other ways of authentication.

The term synchronous has been used to refer to a mode of operation for the MSDS 10 in which the all possible HTML files for a message are generated at the time the message is received. The HTML files may be generated by the central processor 3 or by the application programs 31. When a request for information is then later received by the HTTPD 37, the information has already been generated and the HTTPD 37 only needs to retrieve the information from storage 11 and transmit the information to the user's computer 32. With a synchronous mode of operation, the CGI 35 would be unnecessary.

The MSDS 10 preferably operates according to an asynchronous mode of operation. In an asynchronous mode of operation, information requested by the user may not be available and may have to be generated after the request. The asynchronous mode of operation is preferred since fewer files are generated, thereby reducing the required amount of storage 11. Because the information requested by a user may not be available, some anchors cannot specify the filename, such as "2.html," but will instead contain a command for the file. For instance, an anchor may be defined as `<AHREF="/faxweb/users/2496801/viewpage.cgi?FAX_NUM=1&PAGE=1&VIEW_MODE=FULL">` for causing the CGI 35 to run a viewpage program so that page 1 of facsimile message 1 will be displayed in a full size image. The CGI 35 will generate the requested information when the information has not been generated, otherwise the CGI 35 will retrieve the information and relay the information to the HTTPD 37 for transmission to the user.

With the invention, the MSDS 10 can reliably receive voice, facsimile, and data messages for a plurality of users and can receive more than one message for a user at a single time. The messages are stored by the MSDS 10 and can be retrieved at the user's convenience at any time by connecting to the Internet 30. The Internet World Wide Web 30 is a constantly expanding network that permits the user to retrieve the messages at virtually any location in the world. Since the user only needs to incur a local charge for connecting to the Internet 30, the user can retrieve or review messages at a relatively low cost.

Even for the user's at the office or at home, the MSDS 10 provides a great number of benefits. The user would not need a facsimile machine, voice mail system, or a machine dedicated for receiving data messages. The user also need not worry about losing part of the message or violating the confidential nature of the messages. The user, of course, can still have a facsimile machine and dedicated computer for data messages. The MSDS 10, however, will permit the user to use the telephone company's "call forwarding" feature so

that messages may be transferred to the MSDS 10 at the user's convenience, such as when the user is away from the office.

The software 7 and software 9 are not limited to the exact forms of the flowcharts shown but may be varied to suit the particular hardware embodied by the invention. The software may comprise additional processes not shown or may combine one or more of the processes shown into a single process. Further, the software 7 and 9 may be executed by a single computer, such as a Silicon Graphics Workstation, or may be executed by a larger number of computers.

The facsimile messages preferably undergo signal processing so that the images of the facsimile messages are converted from a two tone black or white image into an image with a varying gray scale. As is known in the art, a gray scale image of a facsimile message provides a better image than simply a black or white image of the message. The signal processing may comprise any suitable standard contrast curve method of processing, such as anti-aliasing or a smoothing filter. The signal processing may occur concurrently with the conversion from TIFF/F to GIF and is preferably performed for both full and reduced size images of the facsimile messages.

Furthermore, the user may be provided with a greater or fewer number of options in displaying or retrieving messages. The options are not limited to the exact forms provided but may permit the user to review or retrieve the messages in other formats. The options may also permit a user to join two or messages into a single message, to delete portions of a message, or to otherwise the contents of the messages. Also, the various menus provided to the user over the telephone may have a greater number of options and the MSDS 10 may accept responses that involve more than just a single DTMF digit.

The specific DTMF digits disclosed in the various menus are only examples and, as will be apparent to those skilled in the art, other digits may be used in their place. For instance, a "9" may be used in the place of a "*" in order to exit the menu or to return to a previous menu. Also, the DTMF digits may be changed in accordance with the user's personal convention. If the user had a previous voice mail system, the user could customize the commands to correspond with the commands used in the previous system in order to provide a smooth transition to the MSDS 10.

The MSDS 10 may restrict a user to only certain types of messages. For instance, a user may want the MSDS 10 to store only facsimile messages in order to reduce costs of using the MSDS 10. In such a situation, the MSDS 10 would perform an additional step of checking that the type of message received for a user is a type of message that the MSDS 10 is authorized to receive on the user's behalf. When the message is an unauthorized type of message, the MSDS 10 may ignore the message entirely or the MSDS 10 may inform the user that someone attempted to send a message to the MSDS 10.

Moreover, the MSDS 10 has been described as having the central processor 3 for handling incoming calls and the Internet Server 10 for interfacing with the Internet 30. The invention may be practiced in various ways other than with two separate processors. For instance, the central processor 3 and the Internet Server 5 may comprise a single computer or workstation for handling the incoming calls and for interfacing with the Internet 30. The MSDS 10 may convert the messages into HTML files prior to storing the messages. Also, the central processor 3 may communicate with the paging system 13 instead of the Internet Server 5.

Additionally, as discussed above, the central processor 3 may comprise a number of microprocessors 27 for handling a large number of DID trunks.

The invention has been described as converting the messages into HTML and transmitting the HTML files over the Internet 30 to the computer 32. The HTML format, however, is only the currently preferred format for exchanging information on the Internet 30 and is actually only one type of a Standard Generalized Mark-Up Language. The invention is therefore not limited to the HTML format but may be practiced with any type of mixed media page layout language that can be used to exchange information on the Internet 30.

SGML is not limited to any specific standard but encompasses numerous dialects and variations in languages. One example of an SGML dialect is virtual reality mark-up language (VRML) which is used to deliver three dimensional images through the Internet. As another example, the computer 32 for accessing the MSDS 10 through the Internet 30 may comprise a handheld device. A handheld device is generally characterized by a small display size, limited input capabilities, limited bandwidth, and limited resources, such as limited amount of memory, processing power, or permanent storage. In view of these limited capabilities, a handheld device markup language (HDML) has been proposed to provide easy access to the Internet 30 for handheld devices. The SGML information transmitted by the MSDS 10 to the computer 32 may therefore comprise HDML information suitable for a handheld device or may comprise VRML.

As another example, Extensible Mark-Up Language (XML) is an abbreviated version of SGML, which makes it easier to define document types and makes it easier for programmers to write programs to handle them. XML omits some more complex and some less-used parts of the standard SGML in return for the benefits of being easier to write applications for, easier to understand, and more suited to delivery and inter-operability over the Web. Because XML is nonetheless a dialect of SGML, the MSDS 10 therefore encompasses the translation of facsimile, voice, and data messages into XML, including all of its dialects and variations, and the delivery of these messages to computers 32 through the Internet 30.

As a further example, the MSDS 10 encompasses the use of "dynamic HTML." "Dynamic HTML" is a term that has been used to describe the combination of HTML, style sheets, and scripts that allows documents to be animated. The Document Object Model (DOM) is a platform-neutral and language neutral interface allowing dynamic access and updating of content, structure, and style of documents. The MSDS 10 may therefore include the use of the DOM and dynamic HTML to deliver dynamic content to the computer 32 through the Internet 30.

The MSDS 10 is also not limited to any particular version or standard of HTTP and thus not to any particular hypertext transfer protocol daemon 37. In general, HTTP is a data access protocol run over TCP and is the basis of the World Wide Web. HTTP began as a generic request-response protocol, designed to accommodate a variety of applications ranging from document exchange and management to searching and forms processing. Through the development of HTTP, the request for extensions and new features to HTTP has exploded; such extensions range from caching, distributed authoring and content negotiation to various remote procedure call mechanisms. By not having a modularized architecture, the price of new features has been an overly complex and incomprehensible protocol. For

instance, a Protocol Extension Protocol (PEP) is an extension mechanism for HTTP designed to address the tension between private agreement and public specification and to accommodate extension of HTTP clients and servers by software components. Multiplexing Protocol (MUX) is another extension that introduces asynchronous messaging support at a layer below HTTP. As a result of these drawbacks of HTTP, a new version of HTTP, namely HTTP-NG, has been proposed and its purpose is to provide a new architecture for the HTTP protocol based on a simple, extensible distributed object-oriented model. HTTP-NG, for instance, provides support for commercial transactions including enhanced security and support for on-line payments. Another version of HTTP, namely S-HTTP, provides secure messaging. The MSDS 10 and the HTTPD 37 may incorporate these versions or other versions of HTTP.

In addition to different versions of HTTP, the HTTPD 37 of the MSDS 10 may operate with other implementations of HTTP. For instance, the W3C's has an implementation of HTTP called "Jigsaw." Jigsaw is an HTTP server entirely written in Java and provides benefits in terms of portability, extensibility, and efficiency. The MSDS 10 may employ Jigsaw or other implementations of HTTP.

With regard to the transmission of messages to the user's computer 32, the MSDS 10 permits the user to sample the voice message or to preview the facsimile message without requiring the MSDS 10 to transmit the entire message to the computer 32. This sampling ability is a significant benefit since the transmission of the entire message would frequently tie up the computer 32 for a rather long period of time. Thus, with the preview or sample feature, the user can determine whether the user needs the message transmitted to the computer 32.

If the user does decide that the entire message needs to be transmitted, as stated above, the user's computer 32 might be receiving the message for a relatively long period of time. After the entire message has been received, the user then has the options of viewing, listening, retrieving, or saving the message. As an alternative, the user's computer may instead indicate the contents of the message to the user as the message is being received.

For instance, with a voice message, the user's computer 32 could send the message to an audio speaker as the message is being received. In this manner, the message would be played in real time and the user would not need to wait until the entire message is received before listening to the message. In order to play the messages in real time, the messages are preferably in the RealAudio (RA) format, which the user can select as the preferred file format for voice messages.

In operation, the MSDS 10 would transmit an HTML file containing an RA file. If the user selects the RA file with the browser on the computer 32, the browser will activate a program for use with RA files. The operations and functioning of this program will be apparent to those skilled in the art and will be available as a separate software package or will be incorporated within a browser program. The RA program will request the RA data file containing the message from the MSDS 10 and, as the RA file is being received at the computer 32, this program will play the message in real time.

The MSDS 10 and the user's computer 32 could also be arranged so that each page or even line of a facsimile message could be displayed as the computer 32 receives the facsimile message. Further, although the transmission of a data message is relatively fast in comparison to a voice or

facsimile message, the computer 32 could also be programmed to permit access to the data message as the message is being received.

The invention has been described as storing and transmitting voice messages. It should be understood that the voice message would probably be the most often type of audio message stored at the MSDS 10. The invention, however, may be used with any type of audio message and is in no way limited to just voice messages.

According to another aspect of the invention, the MSDS 10 may be used as a file repository serving as an archive for a particular user or group of users. As described above, the MSDS 10 may maintain a list of all messages for a particular user which is displayed to the user when the user access his or her mailbox. The MSDS 10 may store all messages, whether they are voice, facsimile, or data, for a user in the database indefinitely. The MSDS 10 may therefore be relied upon by a user to establish the authenticity of a message and the existence or absence of a particular message. Through the MSDS 10, a user can therefore maintain an accurate record of all received email messages, facsimile messages, and data transfers.

In addition to serving as a file depository, the MSDS 10 may also function as a document management tool. As described above with reference to FIG. 2, when the MSDS 10 receives a message, the MSDS 10 updates a database with information on the message. This information includes the type of message, whether it is a facsimile message, voice message, or data message, the time and date at which the message was received, the size of the file, such as in bytes, the telephone number of the caller leaving the message, as well as other information, such as the number of pages of a facsimile message. Because the telephone number called is unique for each user, the information also includes the intended recipient of the message.

An example of a data entry 300 in storage 11 for a message is shown in FIG. 17. The data entry 300 represents the entry for just a single message with each message having a separate data entry 300. Preferably, the data entries 300 are stored in a relational database and may be searched through a structured query language (SQL).

As shown in FIG. 17, the data field 300 for a message may comprise numerous data fields for describing the message. One of these data fields may comprise a field 301 for indicating the name of the person receiving the message. As will be appreciated by those skilled in the art, the person may be identified in numerous ways, such as by a portion of the person's name or by a unique number. Another field 302 in the data entry 300 indicates the type of the document, such as whether the document is a facsimile message, voice message, or data transfer, and fields 303 and 304 respectively indicate the date and time that the message was received by the MSDS 10. The telephone number of the caller is indicated in field 305 while the size of the message, which may be measured in bytes, is indicated in field 306 and the number of pages of the message is indicated in field 307. A document number for uniquely identifying the message is indicated in field 308. As discussed above, the files or messages received for a particular user may be numbered sequentially in the order that they are received by the MSDS 10. The files and messages, however, may be numbered or identified in other ways, such as by a combination of numbers with an identifier for the date when the message was received. Also, the documents number or identifier may be unique for each file or message directed to a user or, alternatively, may be unique for each file or message

directed to a plurality of users, which is advantageous when the MSDS 10 tracks documents for an entire company or other group of users.

In addition to fields 301 to 308, the data entry 300 for a message or file may have other fields 309 for describing or documenting the message or file. The other fields 309, for instance, may be used to identify the type of storage that a message should receive. The messages or files may have different lengths of time that the message is stored before being automatically deleted. The type of storage, such as whether the full text of the message is stored, may also be indicated by field 309. Another example of a trait that may be contained within the other field 309 is security. At times, a user may desire and may be granted access to another person's mailbox, such when the MSDS 10 tracks documents for an entire company. By designating a message or file as secure in field 309, a user may restrict or deny access to that message or file by other users. The other fields 309 may also be used by a user to customize the MSDS 10 according to his or her own desires. For instance, if the user is a company, the company may want to classify messages according to the division at which the message is directed, such as one code for marketing, one for sales, one for engineering, and one for legal.

As another example of a use of one of the other fields 309, a user can input notes in the other field 309. When a user initially receives a data entry 300, the entry 300, for instance, may include data in all fields 301 to 308 except field 309, which has been left blank. The user can then input his or her notes in the other field. An initial data entry 300 may include the field 305 for the caller's telephone number which contains the digits for the calling number. The user, however, may not readily recognize the caller from just reading the telephone number listed in field 305. To more clearly indicate the caller, the user may input notes in field 309 to identify the caller's name. Alternatively, the notes in field 309 may reflect part or all of the contents of the message. The user may receive a large document or message and may input a brief description of the document or message in the field 309. As another example, the recipient of the message may read the message or document and discover that the caller is requesting some service or goods from the recipient, such as a request for certain documents or delivery of a certain quantity of goods. The recipient may read the document or message and place some notes in the field 309 to indicate the type of follow-up service or action that needs to be taken. An assistant to the recipient can then view the notes in field 309 and take appropriate steps to ensure that the requested service or goods are delivered. If the data entry is security protected, one of the other fields 309, as discussed above, may grant the assistant limited access to just the field 309 or may grant more expansive access whereby the assistant can view fields 301 to 309 as well as the actual document or message. The fields 309 may serve various other purposes, as will be apparent to those skilled in the art.

FIG. 18 illustrates a process 320 for using the MSDS 10 for document management purposes. With reference to FIG. 18, a user sends a search request to the MSDS 10 for a particular document or set of documents at step 321. The user may issue this request with the computer 32 by clicking on a link, such as a link to "Search Documents," which may be presented to the user by the MSDS 10 after the user has been granted accesses to his or her mailbox at step 72 shown in FIG. 3. The MSDS 10 may present the user with the option to search the document archives at other times, such as when the user first attempts to access the mailbox at step

62, or when the URL received by the HTTPD 37 from computer 32 points toward the document archives.

In response to this request, the HTTPD 37 sends the user a search query form at step 322 to allow the user to define a desired search. An example of a search query form is shown in FIG. 19. The search query form may include an entry for each of the data fields 301 to 309 in the data entry 300. For instance, the user may input one or more names for a recipient and have the MSDS 10 search for all messages or files directed to just those recipients. The user may also indicate the type of document, such as whether it is a facsimile, voice message or data file. The search query form also has entries for the date or time, which preferably accept ranges of times and dates, and an entry for the telephone number of the caller to the MSDS 10. The search query form may also include an entry for the size of the file or for the number of pages, which is relevant if the message is a facsimile message. The search query form may also include an entry for the document number, which may accept a range of document numbers, and also an entry for another field.

At step 323, the user enters the search parameters in the search query form with computer 32 and returns the information to the MSDS 10 through the Internet 30. The user may define the search about any one data field or may define the search about a combination of two or more data fields. For instance, as reflected in the completed search query form shown in FIG. 20, a user may define a search by designating the document type as a facsimile and the calling number as (404) 249-6801. Once the user has finished defining the search, the user then selects the "SEARCH" link shown at the bottom of the screen whereby the user's computer 32 would send the completed search query form through the Internet 30 to the HTTPD 37 of the MSDS 10.

At step 324, the HTTPD 37 receives the completed search query form and, through CGI 35, invokes one or more of the application programs 31 for performing the desired search for any files or messages falling within the parameters of the search. The results of the search are passed from the application programs 31 through the CGI 35 to the HTTPD 37 and, at step 325, are returned to the user through the Internet 37. Preferably, the MSDS 10 returns the search results in the form of a listing of all files or messages contained within the search parameters, although the MSDS 10 may return the results in other ways.

An example of the search results of the query shown in FIG. 20 is shown in FIG. 21. As discussed above, the parameters of the search were all facsimile messages from telephone number (404) 249-6081. With reference to FIG. 21, this query resulted in three messages being discovered. The first document has a document number 11 and is described as being a facsimile from the designated telephone number to Jane Doe on May 31, 1995, and consists of three pages. This first-listed document is an example of the facsimile shown in FIG. 7. The other two documents respectively correspond to document numbers 243 and 1,002 and are also from the designated telephone number.

At step 326, the user selects the desired file or message from the listing of messages and files. For instance, by clicking on the first listed document, namely document number 11, the computer 32 sends a request to the MSDS 10 for a viewing of that document and, in response, the MSDS 10 provides a viewing of the document according to the user defined preferences. As described above, the user may receive a reduced size image of the first page, a full size image of the first page, reduced size images of all pages, or full size images of all pages of the facsimile message. Thus,

if the user selected the fourth display option as the user defined preference, the MSDS 10 would return an image of the first page of the facsimile, such as the one depicted in FIG. 7.

At step 326, the user may also have the MSDS 10 save the search results. For instance, as shown in FIG. 21, the user may input the name of "CHARLES R. BOBO FACSIMILES" as the name for the search. By clicking on the "SAVE SEARCH AS" link, the name of the search is provided from the computer 32 to the MSDS 10. At the MSDS 10, the HTTPD 37 transfers the information from the computer 32 to the CGI 35 and the CGI 35 invokes an application program 31 to store the results of the search in storage 11 under the designated name. The invoked application program 31 preferably does not store the contents of all messages but rather stores a listing of the search results in the storage 11.

The results of a search may be stored in storage 11 as either a closed search or an open search. If the MSDS 10 saves the results of a search as an open search, then the files or messages in that named search may be updated with recent files or messages falling within the particular search parameters for the search. On the other hand, a closed search is one in which the files or messages in the named search are limited to those existing at the time of the search. For example, if the MSDS 10 saved the search results shown in FIG. 21 as a closed search, then any retrieval of the "CHARLES R. BOBO FACSIMILES" would result in only the three listed documents. If, on the other hand, the search named as the "CHARLES R. BOBO FACSIMILES" was saved by the MSDS 10 as an open search, then the MSDS 10 would reactivate the search query shown in FIG. 20 in response to a request by the computer 32 for that search in order to obtain all facsimile messages from that particular telephone number, including those received after the initial saving of the search results.

With reference to FIG. 19, rather than defining a new search, the user may click on the "STORED SEARCHES" link in order to receive the results of a previously performed search. For example, by clicking on this link, the MSDS 10 may return a listing of searches stored for that particular user, such as the searches shown in FIG. 22. As shown in this figure, the "CHARLES R. BOBO FACSIMILES" is included within the list of stored searches. If the user then selected the "CHARLES R. BOBO FACSIMILES" search, the user may then be presented with the listing of facsimiles shown in FIG. 21, possibly including recent additions to the search group.

With reference to FIG. 19, the MSDS 10 may also provide a user with a link to "RECENT FILES" at step 322. By selecting this link, the MSDS 10 may return a listing of all facsimile, voice, and data messages received with a particular period of time, such as the last month. By placing the "RECENT FILES" link on the search query form rather than in the listing of "STORED SEARCHES," the user can quickly turn to the most recent files and messages. The search query form may contain other such easy-access links, such as a link to the last search performed by the MSDS 10 on behalf of the user.

The messages or files received by the MSDS 10 need not arrive from a third party. In other words, the MSDS 10 may be used as a file repository or as a file manager for documents generated by the user itself. The user may call the designated telephone number for receiving messages and transmit voice messages, data messages, or facsimile messages and have the MSDS 10 document the receipt and

content of these messages. A user may easily use a facsimile machine as a scanner for entering documents into the storage 11 of the MSDS 10.

The MSDS 10 may have applications in addition to those discussed-above with regard to serving as a message deliverer, file repository, and file manager. For instance, the MSDS 10 may perform some additional processing on the incoming calls prior to forwarding them to the user. For voice messages, this processing may involve transcribing the message and then returning the transcribed messages to the user. The MSDS 10 may therefore be viewed as offering secretarial assistance which may be invaluable to small companies or individuals who cannot afford a secretary or even to larger businesses who may need some over-flow assistance. The transcription may be provided by individuals located in any part of the world or may be performed automatically by a speech-to-text recognition software, such as VoiceType from IBM.

Another type of processing that the MSDS 10 may provide is translation services. The incoming call, whether it is a voice, facsimile, or data message, can be converted into SGML and then forwarded first to a translator. Given the reach of the Internet, the translator may be located virtually anywhere in the world and can return the translated document via the Internet to the MSDS 10. The MSDS 10 can notify the user that the translation has been completed through email, voice mail, pager, facsimile, or in other ways. The user would then connect to the Internet and retrieve the translated document. The translation services of the MSDS 10 may also provide transcription of the message, such as with speech-to-text recognition software.

The foregoing description of the preferred embodiments of the invention have been presented only for the purposes of illustration and description. It is not intended to be exhaustive or to limit the invention to the precise form disclosed. Many modifications and variations are possible in light of the above teaching.

The embodiments were chosen and described in order to explain the principles of the invention and their practical application so as to enable others skilled in the art to utilize the invention and various embodiments and with various modifications as are suited to the particular use contemplated. It is intended that the scope of the invention only be limited by the claims appended hereto.

I claim:

1. A network message storage and delivery system, comprising:

means for receiving an incoming call and for detecting an address signal associated with said incoming call, said address signal associated with a user of said message storage and delivery system;

means for receiving a message accompanied with said address signal, said message being in a first file format;

means for converting said message from said first file format to a second file format;

means for storing said message in said second file format in a storage area;

means for receiving a request from said user for said message and for retrieving said message from said storage area; and

means for transmitting a least a portion of said message in said second file format to said user over a transmission link;

wherein said portion of said message is transmitted to said user over the network, said second file format is a

mixed media page layout language and comprises a standard generalized mark-up language.

2. A network message storage and delivery system, comprising:

means for receiving an incoming call and for detecting an address signal associated with said incoming call, said address signal associated with a user of said message storage and delivery system;

means for receiving a message accompanied with said address signal, said message being in a first file format;

means for converting said message from said first file format to a second file format;

means for storing said message in said second file format in a storage area;

means for receiving a request from said user for said message and for retrieving said message from said storage area; and

means for transmitting a least a portion of said message in said second file format to said user over a transmission link;

wherein said portion of said message is transmitted to said user over the network, said second file format is a mixed media page layout language, and said network comprises the Internet.

3. A network message storage and delivery system, comprising:

a central processor for receiving an incoming call, for detecting an address signal on said incoming call, for detecting a message on said incoming call, and for placing said message in a storage area, said address signal being associated with a user of said network message storage and delivery system;

a network server for receiving said message from said storage area, for converting said message into a mixed media page layout language, and for placing said message in said storage area;

wherein when said network server receives a request from said user over said network, said network server transmits at least a portion of said message over said network to said user over a transmission link and wherein said network comprises the Internet and said network server comprises an Internet server.

4. A method of storing and delivering a message for a user, comprising the steps of:

receiving an incoming call and detecting an address signal associated with said incoming call, said address signal associated with a user;

receiving a message associated with said address signal, said message being in a first file format;

converting said message from said first file format to a second file format;

storing said message in said second file format in a storage area;

receiving a request from said user for said message and retrieving said message from said storage area; and

transmitting at least a portion of said message in said second file format to said user over a transmission link;

wherein said step of transmitting occurs over a network, said step of converting said message converts said message into a mixed media page layout language, and said step of transmitting occurs over the Internet.

* * * * *



US006658463B1

(12) **United States Patent**
Dillon et al.

(10) **Patent No.:** **US 6,658,463 B1**
(45) **Date of Patent:** **Dec. 2, 2003**

(54) **SATELLITE MULTICAST PERFORMANCE
ENHANCING MULTICAST HTTP PROXY
SYSTEM AND METHOD**

(75) Inventors: **Douglas M. Dillon**, Gaithersburg, MD
(US); **T. Paul Gaske**, Rockville, MD
(US)

(73) Assignee: **Hughes Electronics Corporation**, El
Segundo, CA (US)

(*) Notice: Subject to any disclaimer, the term of this
patent is extended or adjusted under 35
U.S.C. 154(b) by 0 days.

(21) Appl. No.: **09/498,936**

(22) Filed: **Feb. 4, 2000**

Related U.S. Application Data

(60) Provisional application No. 60/138,496, filed on Jun. 10,
1999.

(51) Int. Cl.⁷ **G06F 15/16**

(52) U.S. Cl. **709/219; 709/217; 709/203;**
709/228

(58) Field of Search **709/217, 219,**
709/202, 203, 204; 707/1, 8, 100, 200,
201

(56) **References Cited**

U.S. PATENT DOCUMENTS

6,055,364 A * 4/2000 Speakman et al. 709/229
6,085,193 A * 7/2000 Malkin et al. 709/200
6,128,655 A * 10/2000 Fields et al. 709/219
6,463,447 B2 * 10/2002 Marks et al. 709/228

OTHER PUBLICATIONS

"Transparent Power-on Control of Token Ring and Token
Bus File Server Computers", IBM Technical Disclosure
Bulletin, Mar. 1995, vol. 38, Issue 3, pp. 55-56.*

* cited by examiner

Primary Examiner—David Wiley

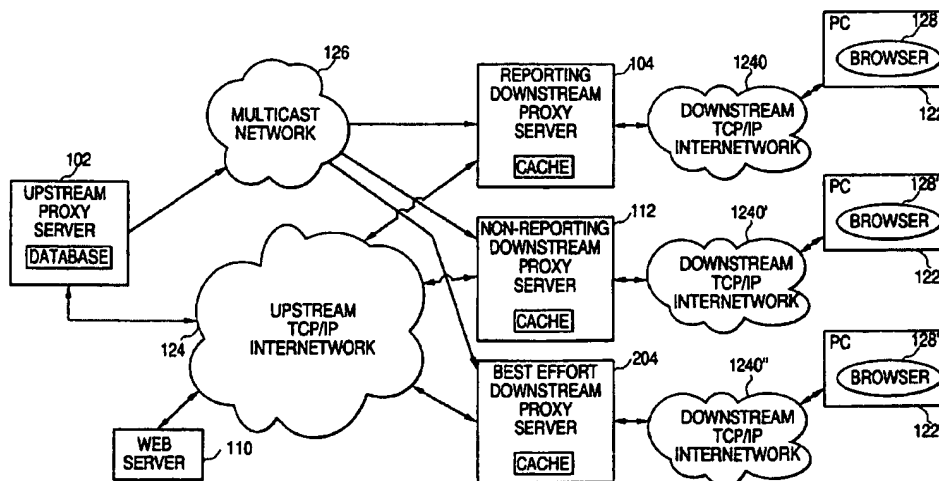
Assistant Examiner—Joseph E. Avellino

(74) *Attorney, Agent, or Firm*—John T. Whelan; Michael
Sales

(57) **ABSTRACT**

A communication system including an upstream proxy server and two reporting downstream proxy servers, where the upstream proxy server is capable of multicasting a uniform resource locator (URL) to the reporting downstream proxy servers, the reporting downstream proxy servers interact with the upstream proxy server to resolve cache misses and the upstream proxy servers returns a resolution to a cache miss via multicast. A downstream proxy server which filters multicast transmissions of URLs and stores a subset of the URLs for subsequent transmission, where relative popularity is used to determine whether to store a multicast URL. An upstream proxy server capable of multicasting URLs to reporting downstream proxy servers, where the upstream proxy server interacts with the two reporting downstream proxy servers to resolve cache misses and the upstream proxy server returns a resolution to the cache misses via multicast. A proxy server protocol which includes a transaction request further including a request header, request content, and a request extension that supports multicast hit reporting and a transaction response further including a response header, response content, and a response extension which supports multicast cache pre-loading. A transaction response header which includes a popularity field indicating the popularity of a global name with respect to other global names and an expiration field indicating an expiration of the global name.

20 Claims, 12 Drawing Sheets



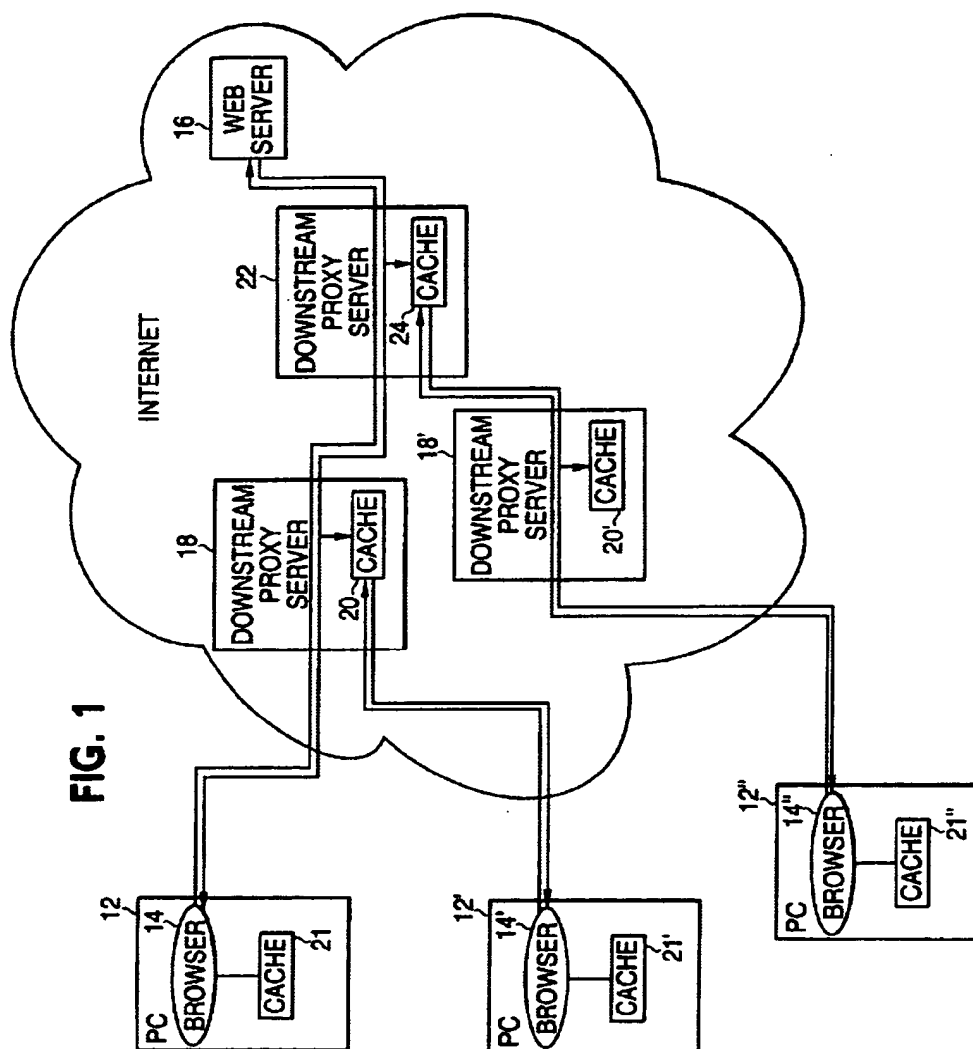
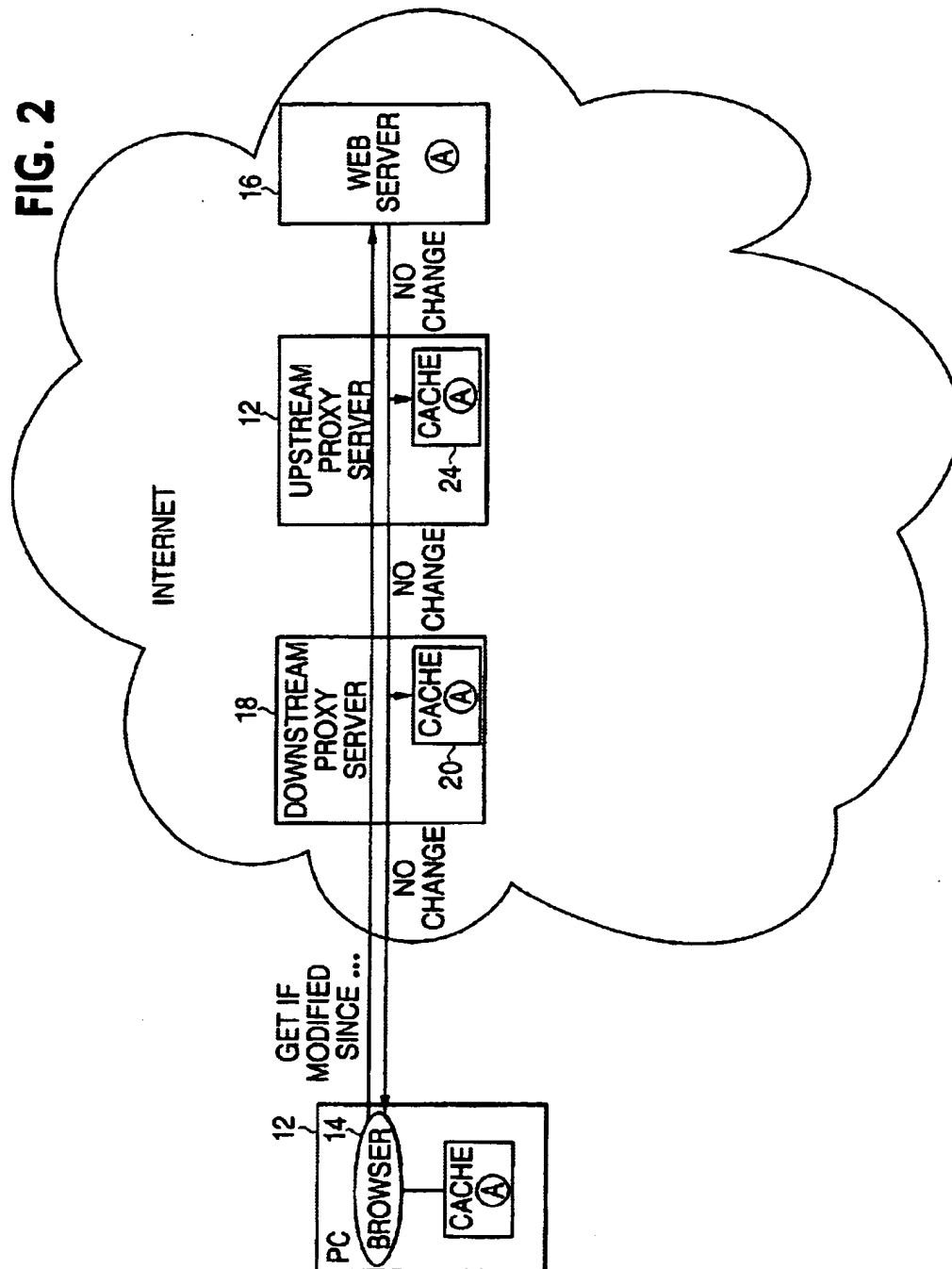
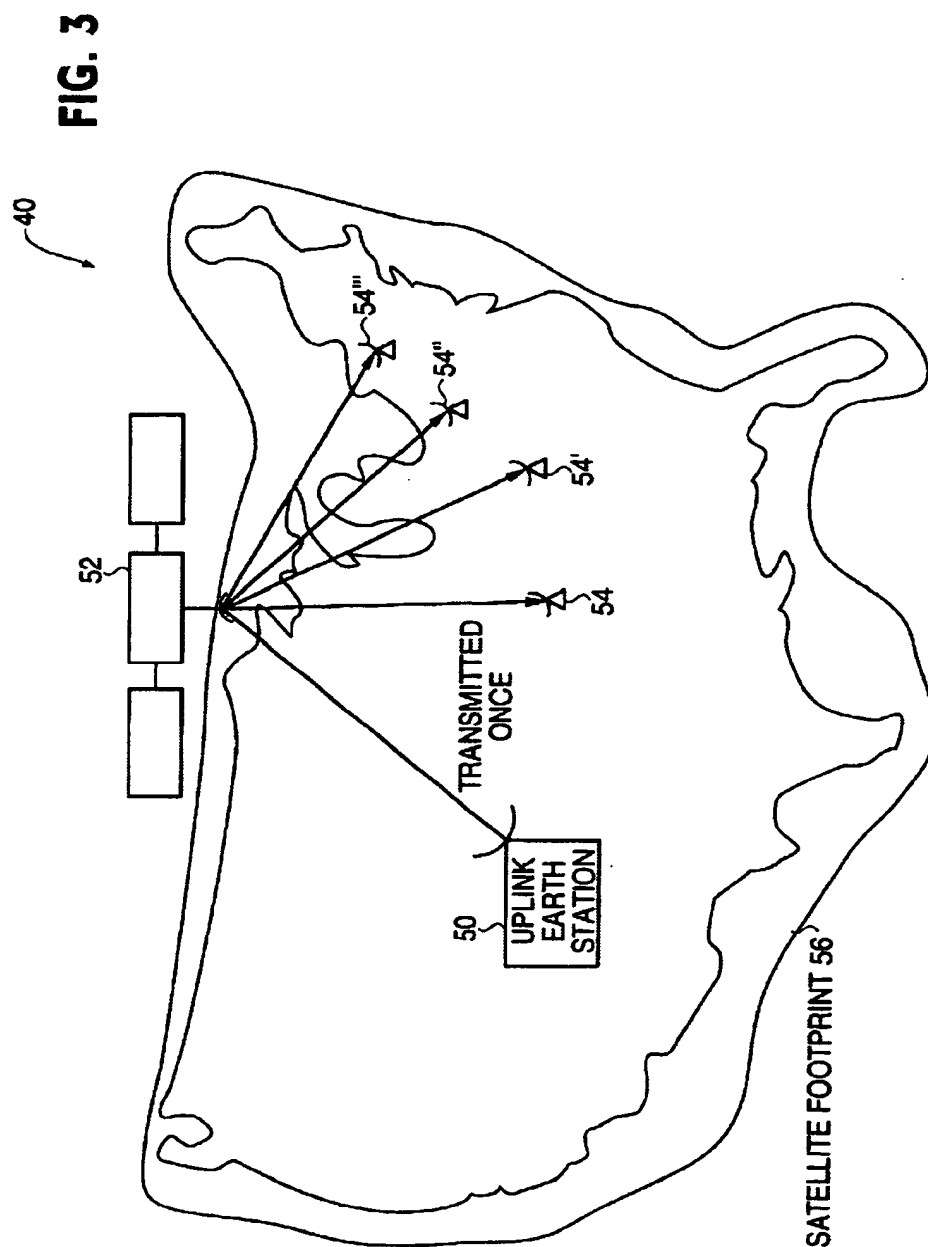
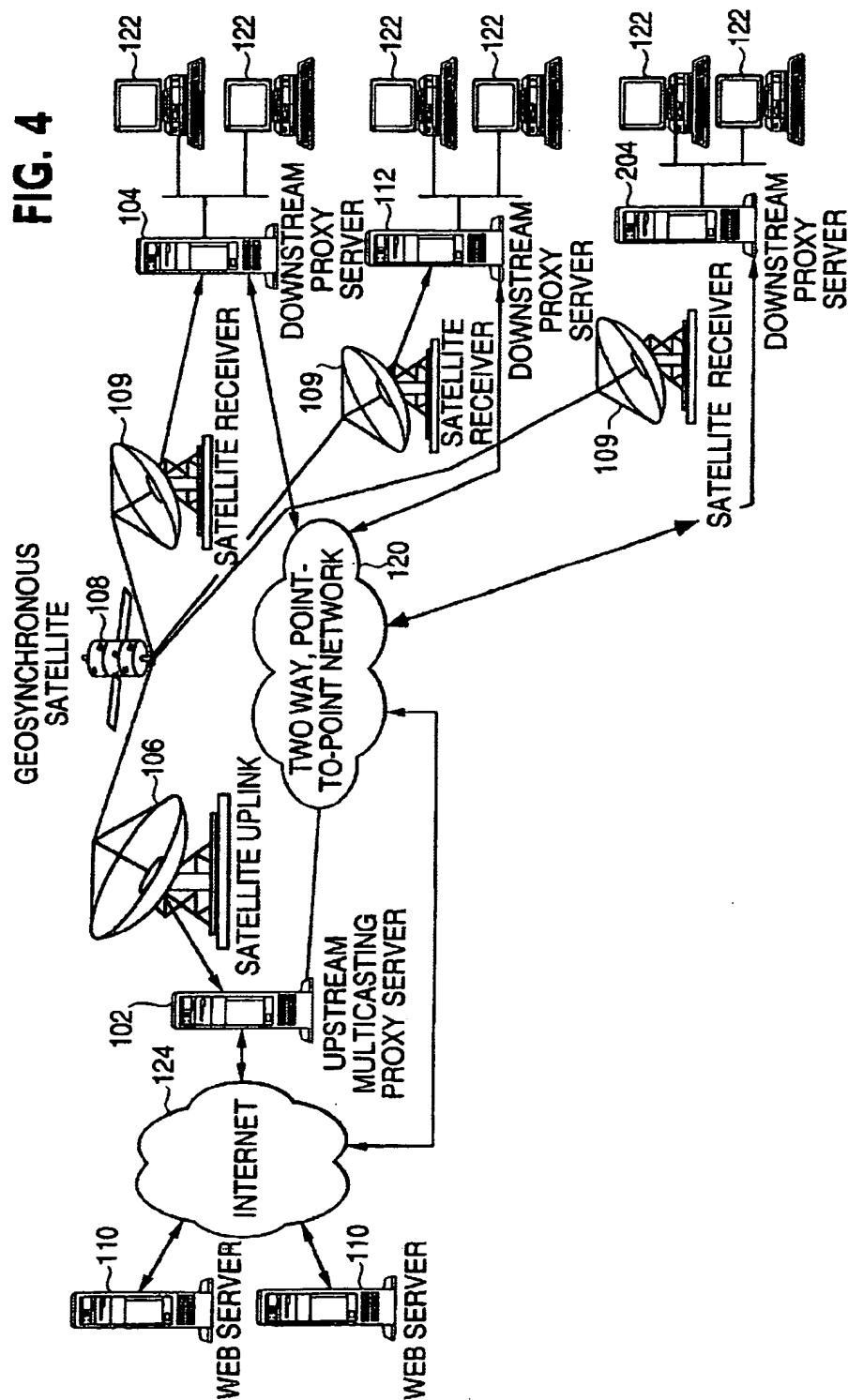


FIG. 2





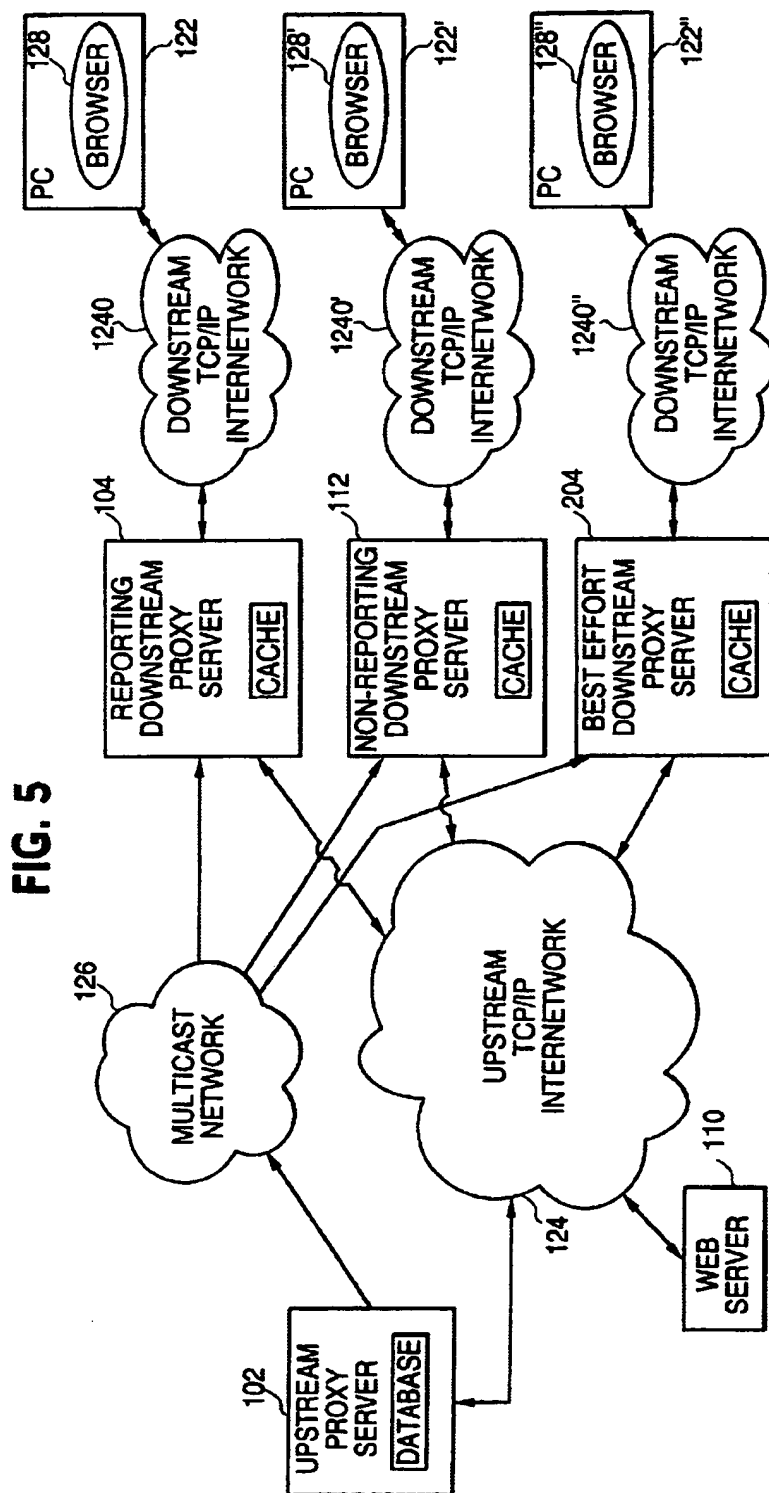


FIG. 5a

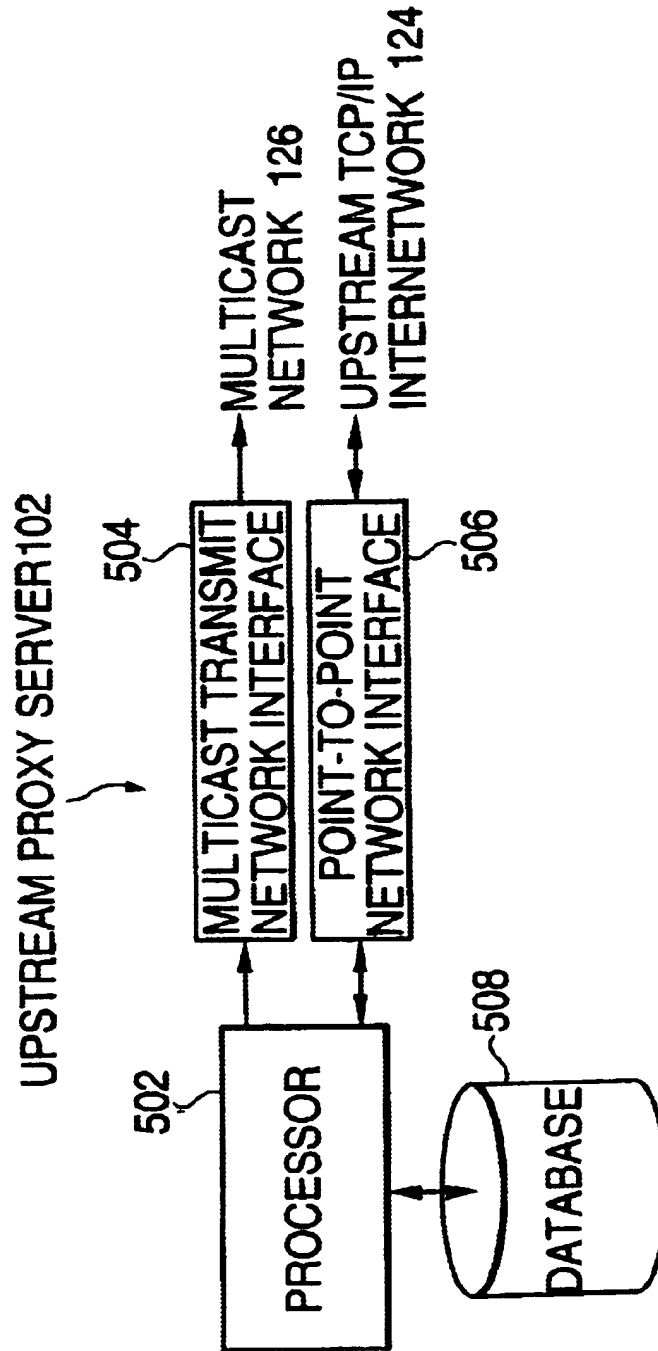


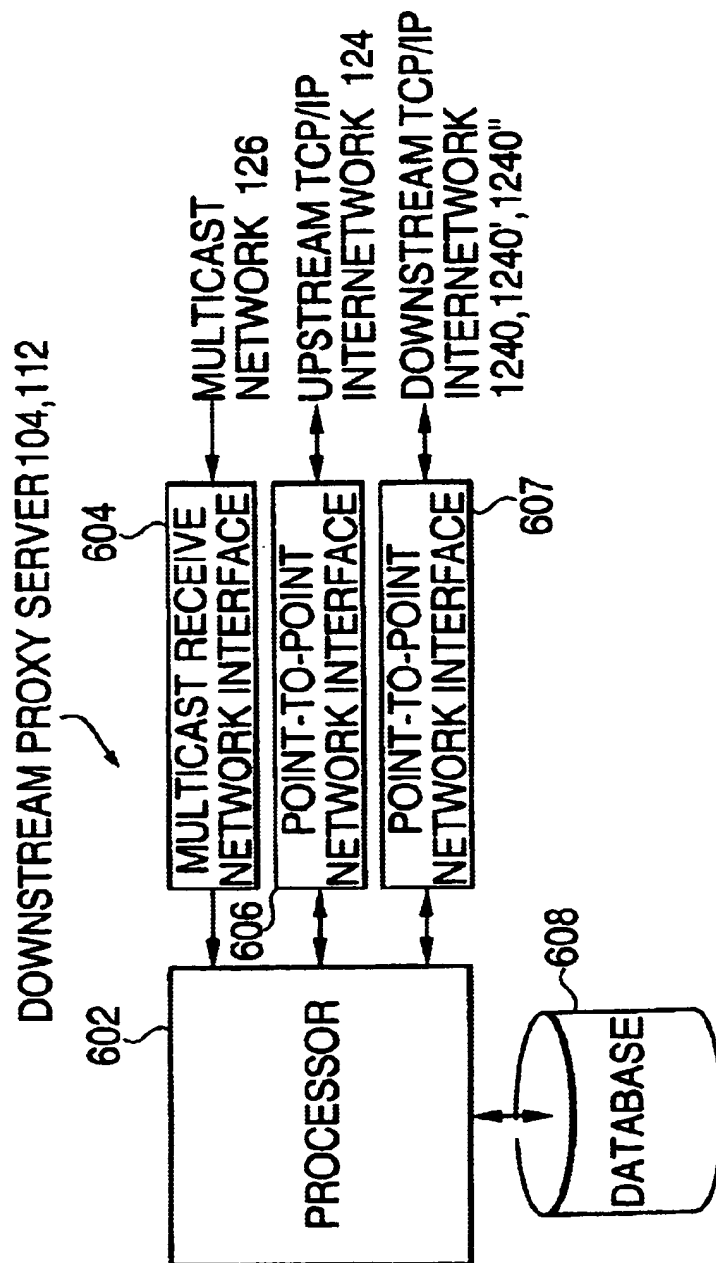
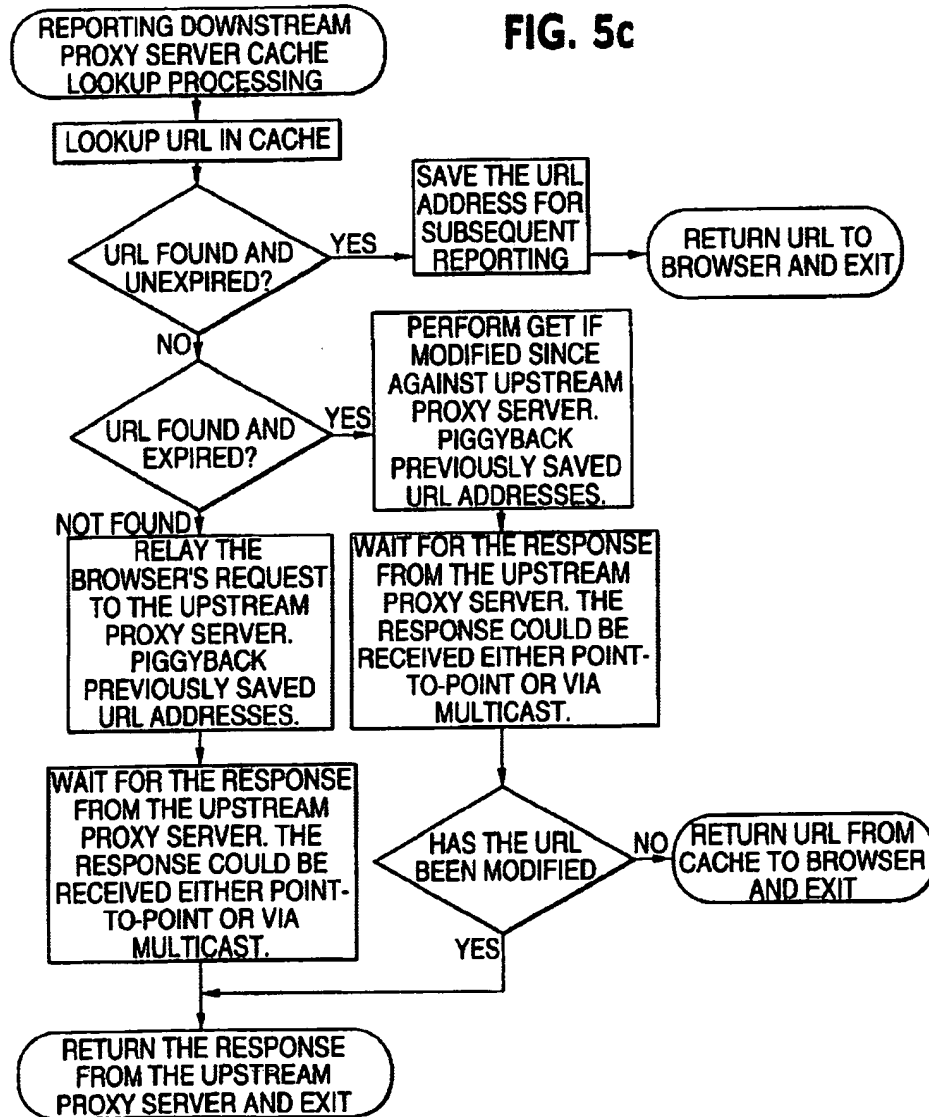
FIG. 5b

FIG. 5c



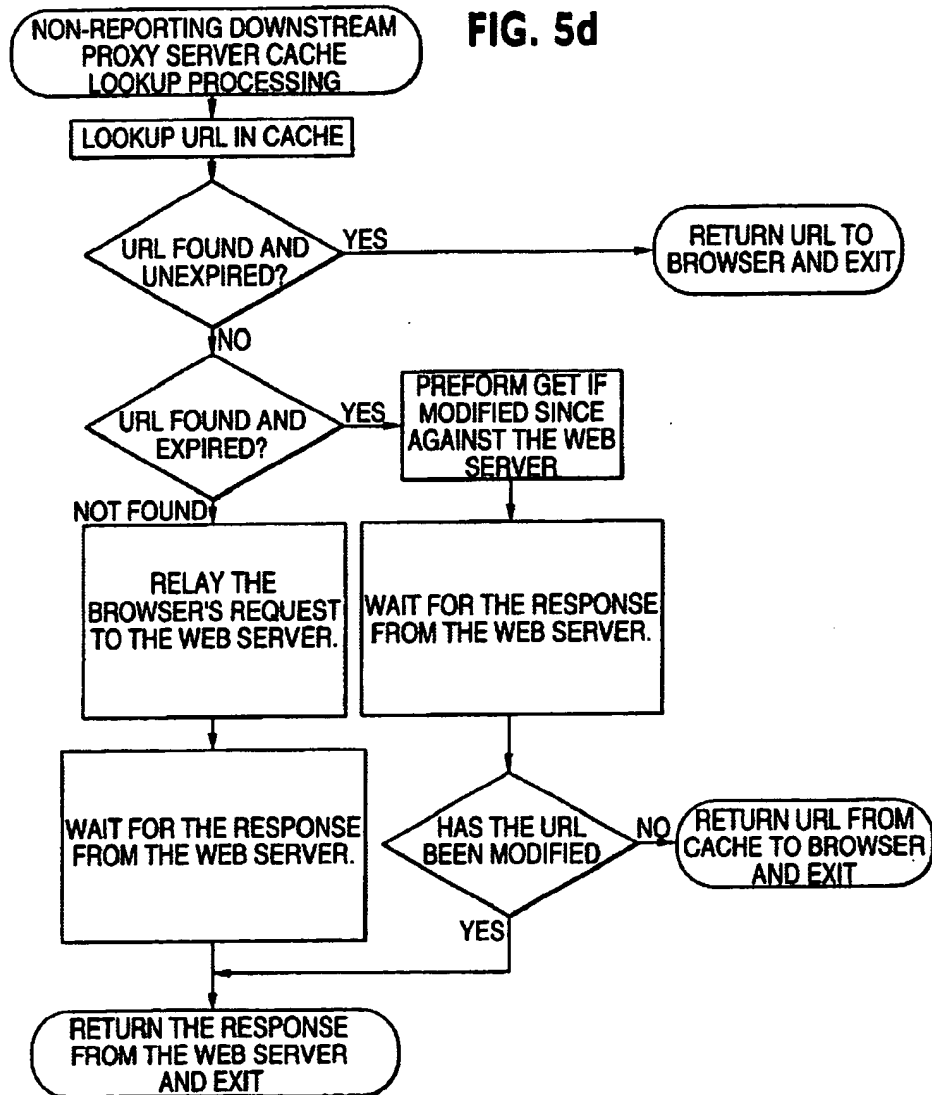
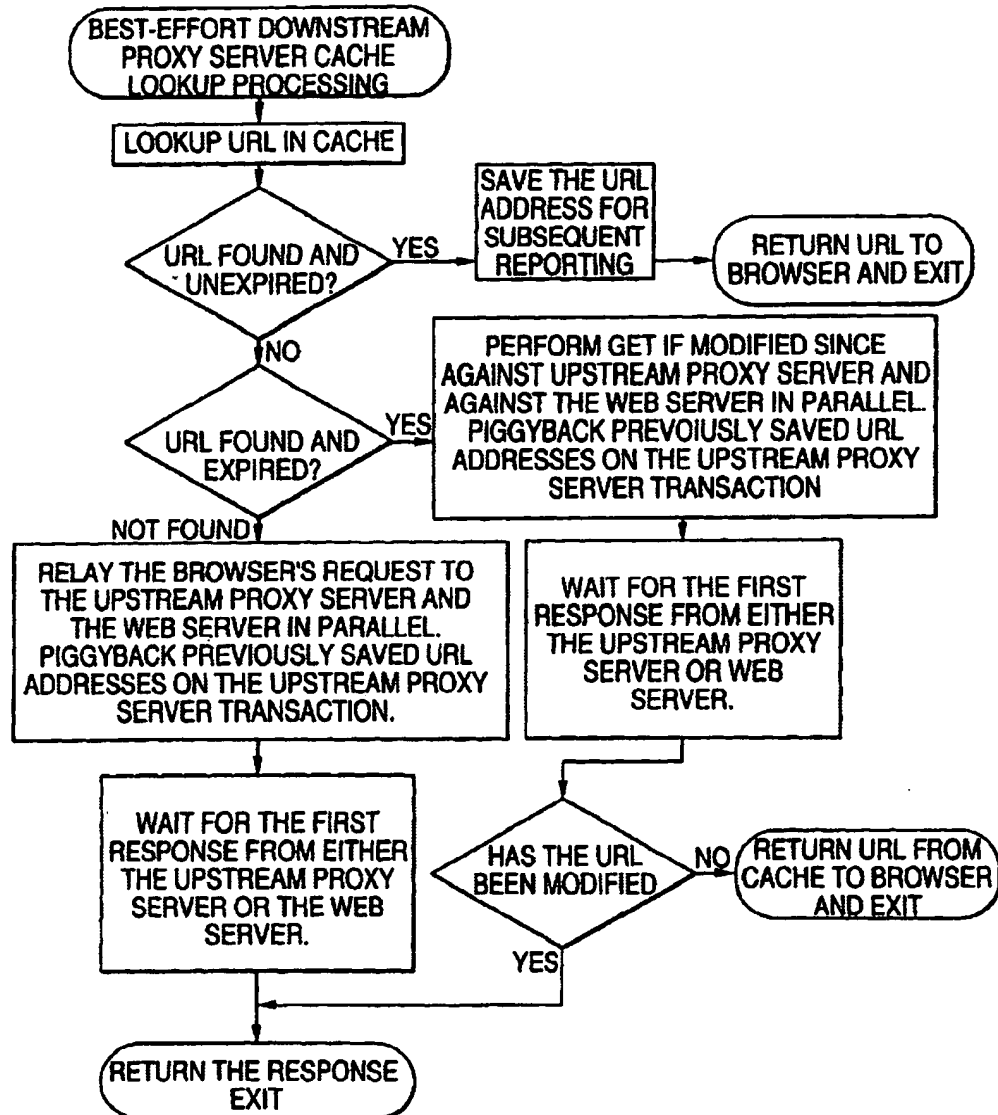


FIG. 5e



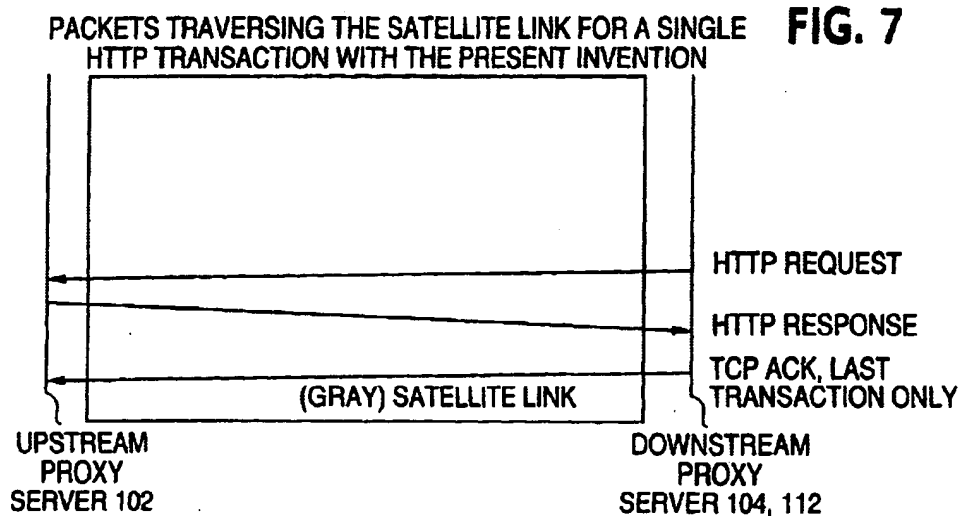
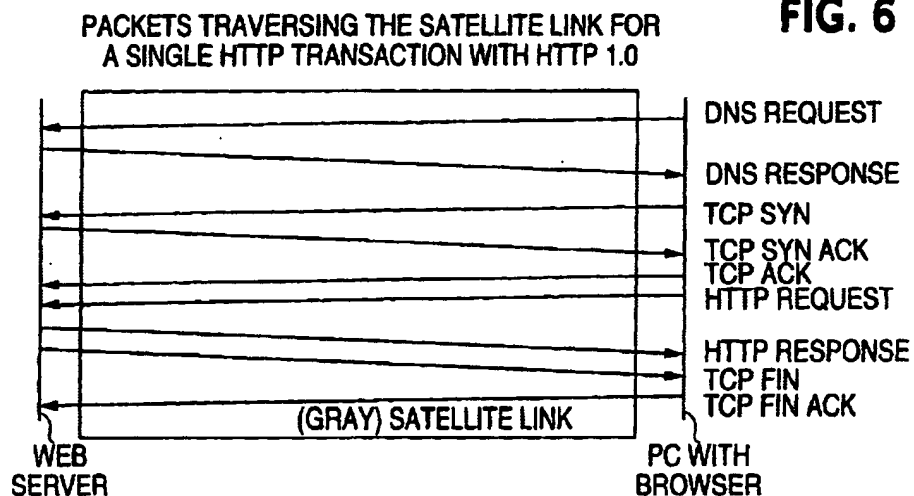


FIG. 8

GET <http://www.yahoo.com/HTTP/1.0>
PROXY-CONNECTION: KEEP-ALIVE
USER-AGENT: MOZILLA/4.61 [EN] (WinNT;1)
HOST: www.yahoo.com
ACCEPT: IMAGE/GIF, IMAGE/X-BITMAP, IMAGE/PEG, IMAGE/PJPEG, IMAGE/PNG,1
ACCEPT-ENCODING: GZIP
ACCEPT-LANGUAGE: EN
ACCEPT-CHARSET: ISO-8859-1, *, utf-8
COOKIE: B 31c61vhar6s

FIG. 9

HTTP/1.0 200 OK
PROXY-CONNECTION: KEEP-ALIVE
CONNECTION: KEEP-ALIVE
CONTENT-LENGTH: 11831
CONTENT-TYPE: text/html

1

SATELLITE MULTICAST PERFORMANCE ENHANCING MULTICAST HTTP PROXY SYSTEM AND METHOD

RELATED APPLICATIONS

This application is based on and claims benefit from provisional application entitled "Satellite Multicast Performance Enhancing Multicast HTTP Proxy System and Method" which was filed on Jun. 10, 1999, and respectively accorded Serial No. 60/138,496.

1. BACKGROUND OF THE INVENTION

1.1 Field of the Invention

The present invention relates generally to the distribution of World Wide Web content over a geosynchronous satellite communications network, and in particular, to satellite communications networks having an outbound high-speed, continuous channel carrying packetized data and either a satellite inbound channel or a terrestrial inbound channel, such as a dialup connection to the Internet.

1.2 Description of related Art

1.2.1 Caching HTTP Proxy Servers

The most popular method for distributing multimedia information is the Internet's World Wide Web. The World Wide Web can be considered to be a set of network accessible information resources. In the World Wide Web, many Web Servers and Web Browsers are connected to the Internet via the TCP/IP protocols and the Web Browsers request web pages and graphics and other multimedia content via the Hypertext Transfer Protocol (HTTP).

The World Wide Web is founded on three basic ideas:

1. A global naming scheme for resources—that is, Uniform Resource Locators (URLs).
2. Protocols for accessing named resources—the most common of which is the Hypertext Transfer Protocol (HTTP).
3. Hypertext—the ability to embed links to other resources which is typically done according to the Hypertext Markup Language (HTML).

Web pages are formatted according to the Hypertext Markup Language (HTML) standard which provides for the display of high-quality text (including control over the location, size, color and font for the text), the display of graphics within the page and the "linking" from one page to another, possibly stored on a different web server. Each HTML document, graphic image, video clip or other individual piece of content is identified, that is, addressed, by an Internet address, referred to as a Uniform Resource Locator (URL). In the context of this invention, a "URL" may refer to an address of an individual piece of web content (HTML document, image, sound-clip, video-clip, etc.) or the individual piece of content addressed by the URL. When a distinction is required, the term "URL address" refers to the URL itself while the terms "URL content" or "URL object" refers to the content addressed by the URL.

A web browser may be configured to either access URLs directly from a web server or from an HTTP proxy server. An HTTP proxy server acts as an intermediary between one or more browsers and many web servers. A web browser requests a URL from the proxy server which in turn "gets" the URL from the addressed web server. An HTTP proxy itself may be configured to either access URLs directly from a web server or from another HTTP proxy server. When a proxy server sends a request to another proxy server the proxy server processing the request is referred to as being

2

upstream (that is, closer to the web server). When a proxy server receives a request from another proxy server, the requesting proxy server is referred to as being downstream, that is, farther from the Web Server.

FIG. 1 illustrates a system in which one of a plurality of browsers accesses a web server via upstream and downstream proxy servers with an HTTP GET command. In particular, a plurality of PCs 12, each including a browser 14, output a GET command to web server 16, in order to access the URL "A". Assuming PC 12 and browser 14 make the first request, the GET command is passed to downstream proxy server 18. Since this is the first request for URL "A", the downstream proxy server 18 does not have URL "A" in its cache 20. As a result, the downstream proxy server 18 also issues a GET URL "A" command to upstream proxy server 22. Since this is also the first request to upstream proxy server 22 for the URL "A", the upstream proxy server 22 also does not have URL "A" in its cache 24. Therefore, the upstream proxy server 22 issues a GET URL "A" command directly to the web server 16. The web server 16 services this request and provides the upstream proxy server 22 with the desired information, which is then stored in the cache 24. The upstream proxy server 22 passes the desired information to the downstream proxy server 18, which also stores the desired information in its cache 20. Finally, the downstream proxy server 18 passes the desired information to the originating requestor's browser 14 at PC 12, which also stores the desired information in its cache 21.

Subsequently, PC 12', via its browser 14', also desires the information at URL "A". PC 12' issues a GET URL "A" command to downstream proxy server 18'. At this time, downstream proxy server 18' has the desired information in its cache 20 and provides the information directly to PC 12' without requesting additional information from either the upstream proxy server 22 or the web server 16. Similarly, if PC 12'', via its browser 14'', also desires the information at URL "A", PC 12'' issues a GET URL "A" command to downstream proxy server 18'. However, since downstream proxy server 18' does not have the information for URL "A" stored in its cache 20', the downstream proxy server 18' must access the upstream proxy server 22 and its cache 24, in order to supply the desired information to PC 12''. However, the upstream proxy server 22 does not have to access the web server 16, because the desired information is stored in its cache 24.

As described above, a caching HTTP proxy server, such as downstream proxy servers 18, 18' and upstream proxy server 22 store (cache) some URLs. Normally, a caching proxy server stores the most frequently accessed URLs. When a web server delivers a URL, it may deliver along with the URL an indication of whether the URL should not be cached and an indication of when the URL was last modified. As described in conjunction with FIG. 1, the URLs stored by a caching proxy server are typically URLs obtained on behalf of a browser or downstream proxy server. A caching HTTP proxy server satisfies a request for a URL, when possible, by returning a stored URL. The HTTP protocol also supports a GET IF MODIFIED SINCE request wherein a web server (or a proxy server) either responds with a status code indicating that the URL has not changed or with the URL content if the URL has changed since the requested date and time.

FIG. 2 illustrates a browser executing a GET IF MODIFIED SINCE command from web server 16. As illustrated in FIG. 2, the PC 12, including browser 14, has already requested URL "A" once and has URL "A" stored in its cache 21. PC 12 now wants to know if the information stored

3

at URL "A" has been updated since the time it was last requested. As a result, the browser 14 issues a GET A IF MODIFIED SINCE the last time "A" was obtained. Assuming that URL "A" was obtained at 11:30 a.m. on Jul. 13, 1999, browser 14 issues a GET A IF MODIFIED SINCE Jul. 15, 1999 at 11:30 a.m. request. This request goes to downstream proxy server 18. If downstream proxy server 18 has received an updated version of URL "A" since Jul. 15, 1999 at 11:30 a.m., downstream proxy server 18 will supply the new URL "A" information to the browser 14. If not, the downstream proxy server 18 will issue a GET IF MODIFIED SINCE command to upstream proxy server 22. If upstream proxy server 22 has received an updated URL "A" since Jul. 15, 1999 at 11:30 a.m., upstream proxy server 22 will pass the new URL "A" to the downstream proxy server 18. If not, the upstream proxy server 22 will issue a GET A IF MODIFIED SINCE command to the web server 16. If URL "A" has not changed since Jul. 15, 1999 at 11:30 a.m., web server 16 will issue a NO CHANGE response to the upstream proxy server 22. In this way, bandwidth and processing time are saved, since if the URL "A" has not been modified since the last request, the entire contents of URL "A" need not be transferred between web browser 14, downstream proxy server 18, upstream proxy server 22, and the web server 16, only an indication that there has been no change need be exchanged.

Caching proxy servers offer both reduced network utilization and reduced response time when they are able to satisfy requests with cached URLs. Much research has been done attempting to arrive at a near-optimal caching policy, that is, determining when a caching proxy server should store URLs, delete URLs and satisfy requests from the cache both with and without doing a GET IF MODIFIED SINCE request against the web server. Caching proxy servers are available commercially from several companies including Microsoft, Netscape, Network Appliance and Cache Flow. 1.2.2 Satellite Multicast Networks

Typical geosynchronous satellites relay a signal from a single uplink earth station to any number of receivers under the "foot print" of the satellite. FIG. 3 illustrates a typical satellite system 40. The satellite system 40 includes an uplink earth station 50, a satellite 52, and receiving terminals 54, 54', 54", 54'''. The satellite system 40 covers a footprint 56, which in the example in FIG. 3, is the continental United States. The footprint 56 typically covers an entire country or continent. Multicast data is data which is addressed to multiple receiving terminals 54. When the signal is carrying digital, packetized data, a geosynchronous satellite 52 is an excellent mechanism for carrying multicast data as a multicast packet need only be transmitted once to be received by any number of terminals 54. Such a signal, by carrying both unicast and multicast packets can support both normal point-to-point and multicast applications. Satellite multicast data systems are typically engineered with Forward Error Correcting (FEC) coding in such a way that the system is quasi-error free, that is, under normal weather conditions packets are hardly ever dropped.

The Internet Protocol (IP) is the most commonly used mechanism for carrying multicast data. Satellite networks capable of carrying IP Multicast data include Hughes Network System's Personal Earth Station VSAT system, Hughes Network System's DirecPC™ system as well as other systems by companies such as Gilat, Loral Cyberstar and Media4.

VSAT systems, such as the Personal Earth Station by Hughes Network Systems, use a satellite return channel to support two-way communication, when needed. For World

4

Wide Web access, a terminal using a VSAT system sends HTTP requests to the Internet by means of the VSAT's inbound channel and receive HTTP responses via the outbound satellite channel. Other systems, such as DirecPC's™ Turbo Internet, use dialup modem. (as well as other non-satellite media) to send HTTP requests into the Internet and receive responses either via the outbound satellite channel or via the dialup modem connection. Satellite networks often have a longer latency than many terrestrial networks. For example, the round trip delay on a VSAT is typically 1.5 seconds while the round trip delay of dialup Internet access is typically only 0.4 seconds. This difference in latency is multiplied in the case of typical web browsing in that multiple round trips are required for each web page. This places web browsing via satellite at a distinct disadvantage relative to many terrestrial networks. The present invention provides a major reduction in this disadvantage and as such greatly increases the value of web browsing via satellite.

2. SUMMARY OF THE INVENTION

The present invention is directed to a communication network having an outbound high-speed channel carrying packetized data and either a satellite inbound channel or a terrestrial inbound channel, such as a dial-up connection to the internet. The communication network includes at least one upstream proxy server and at least two reporting downstream proxy servers, where the at least one upstream proxy server is capable of multicasting URLs to the at least two reporting downstream proxy servers. The at least two reporting downstream proxy servers interact with the at least one upstream proxy server to resolve cache misses and the at least one upstream proxy server returns at least one resolution to the cache misses via multicast. The proxy servers included in the communication system may include reporting proxy servers, non-reporting proxy servers, and best effort proxy servers. A reporting downstream proxy server interacts with an upstream proxy server to satisfy a cache miss. A non-reporting downstream proxy server interacts with a web server to satisfy a cache miss. A best effort downstream proxy server requests a cache-miss URL from both the upstream proxy server and the web server.

In one embodiment, the downstream proxy server filters multicast transmissions of URLs and stores the subset of the URLs for subsequent transmission where relative popularity is used to determine whether to store a multicast URL. In one embodiment, the upstream proxy server is capable of multicasting URLs to at least two reporting downstream proxy servers, the upstream proxy server interacts with the two reporting downstream proxy servers to resolve cache misses and the upstream proxy server returns at least one resolution to the cache misses via multicast.

In another embodiment, the downstream reporting proxy server includes a data base and a processor for receiving entries sent by an upstream proxy server, for filtering unpopular entries, keeping popular entries in the database, deleting previously stored entries from the data base, expiring previously stored entries from the data base, or reporting new entries to the upstream proxy server.

As described above, the communication system lowers user response time, lowers network utilization, and reduces the resources required by an HTTP proxy server.

In other embodiments, the present invention is directed to a proxy protocol which performs transaction multiplexing which prevents a single stalled request from stalling other requests, performs homogenized content compression which intelligently compresses HTTP request and response headers

5

and performs request batching so that nearly simultaneously received requests are sent in a single TCP segment, in order to reduce the number of required inbound packets.

3. BRIEF DESCRIPTION OF THE DRAWINGS

FIG. 1 illustrates a conventional system, including browsers, web servers, upstream and downstream proxy servers, and the execution of a GET COMMAND.

FIG. 2 illustrates a conventional system, including browsers, web servers, upstream and downstream proxy servers, and the execution of a GET IF MODIFIED SINCE COMMAND.

FIG. 3 illustrates a conventional satellite system.

FIG. 4 illustrates a communication system in one embodiment of the preferred invention.

FIG. 5 illustrates a communication system in another embodiment of the present invention.

FIG. 5a illustrates an upstream proxy server in one embodiment of the present invention.

FIG. 5b illustrates a downstream proxy server in one embodiment of the present invention.

FIG. 5c illustrates the cache lookup processing performed by a reporting downstream proxy server in one embodiment of the present invention.

FIG. 5d illustrates the cache lookup processing performed by a non-reporting downstream proxy server in one embodiment of the present invention.

FIG. 5e illustrates the cache lookup processing performed by a best-effort downstream proxy server in one embodiment of the present invention.

FIG. 6 illustrates the TCP/IP packets which traverse the communication link for a single HTTP transaction without the benefit of the present invention.

FIG. 7 illustrates the TCP/IP packets which traverse the network medium for a single HTTP transaction with the benefit of one embodiment of the present invention.

FIG. 8 illustrates an HTTP request in one embodiment of the present invention.

FIG. 9 illustrates an HTTP response in one embodiment of the present invention.

4. DETAILED DESCRIPTION OF THE PREFERRED EMBODIMENT

4.1 INTRODUCTION TO THE INVENTION

While there has been some work using satellite multicast to preload an HTTP proxy server cache, the present invention includes several innovations which increase (often dramatically) the utility of such a system when a single user or small number of users access the network through a single satellite multicast receiving proxy. These innovations provide:

1. lower user response time;
2. lower network utilization; and

As depicted by FIG. 4, in one exemplary embodiment, the present invention includes an upstream, multicasting proxy server 102 and multiple downstream multicast receiving proxy servers 104, 112, 204. The upstream proxy server 102 multicasts web content to the downstream proxy servers 104, 112, 204 by means of a one-way capable multicast network such as a geosynchronous satellite broadcast. The one-way capable multicast network includes the satellite uplink 106, the geosynchronous satellite 108, and satellite receiver 109. A subset of the downstream proxy servers 104,

6

referred to as "reporting proxy servers" interact with the upstream proxy server 102 by means of a two-way point-to-point capable network 120, examples of which include a dialup access internet network and satellite VSAT systems carrying interactive TCP/IP. Other downstream proxy servers 112 interact with web servers 110 for cache misses without going through the upstream proxy server 102. These proxy servers 112 are referred to as non-reporting proxy servers. Yet another class of downstream proxy servers 204 interact with both the upstream proxy server 102 and the web server 110 in parallel and passes the HTTP response back to a web browser of PC 122 from whichever responds first. These proxy servers are referred to as best effort proxy servers. In some cases, such as a VSAT system, the multicast network and the two-way point-to-point network may be a single integrated network. In other cases, they may be separate networks.

4.2 PRIOR ART SATELLITE MULTICAST CACHING PROXY SERVER SYSTEMS

There are two classes of known prior art satellite multicast caching proxy server systems.

Multicast push systems, such as, the DirecPC™ TurboWebCast service which is available with DirecPC™ sold by Hughes Network Systems, allow users to subscribe to a set of web channels, where a channel is typically a portion of a web site. The content of the channel is multicast file-transferred to subscribing users and a proxy server on the subscribing user's PC allows the user to access data from the cache offline, without any two-way connection to the Internet.

Large-scale multicast caching systems, such as the system developed by SkyCache multicast content to caching proxies loaded in cable modem head ends and Internet Service Provider points of presence (POPs).

As will be clear from the discussion that follows, the present invention distinguishes from the prior art in several ways, namely:

1. Unlike a multicast push system, the present invention reduces the response time and network utilization experienced by users without requiring any explicit subscription to content by the user and without requiring the preparation and maintenance of channel definitions by the satellite uplink.
2. Unlike large-scale multicast caching systems, the present invention includes novel filtering of multicast URLs to minimize the processing associated with filtering multicast URLs.
3. Unlike large-scale multicast caching systems, the present invention operates correctly and effectively without requiring the continuous operation of a downstream proxy.
4. Unlike large-scale multicast caching systems, the present invention often uses the multicast channel to send URLs in response to a downstream proxy server request thereby reducing the network loading on the point-to-point network connecting the downstream proxy to the upstream proxy. The point-to-point network and the multicast network are often a single, integrated satellite network. When this is the case, multicasting the URL consumes no more network capacity than transmitting it point-to-point while offering the benefit of possibly eliminating future transmissions of the URL by preloading the URL into other receiver's caches.
5. Unlike large-scale multicast caching systems, the present invention allows the downstream proxy server

7

to automatically cease the processing of multicast traffic when a user is actively using a PC that the downstream proxy server is running on.

6. Unlike large-scale multicast caching systems, the downstream proxy servers pass usage information to the upstream proxy server. The upstream proxy server factors this usage information into its decision whether to multicast URLs.

4.3 SYSTEM OVERVIEW

The term TCP/IP in the context of this invention refers to either the current version of TCP/IP (IP version 4) or the next generations (for example, IP version 6). The basics of TCP/IP internetworking as known by one of ordinary skill in the art, can be found in "Internetworking with TCP/IP Volume 1" by Douglas Comer.

As illustrated in FIG. 5, the present invention allows web browsers 128, 128', 128" to access multiple web servers 110, only one such web server being depicted. The web servers 110 and the upstream proxy server 102 are connected to a TCP/IP internetwork 124 referred to as the upstream internetwork. The upstream proxy server 102 is able to multicast to the downstream proxy servers 104, 112, 204 by the multicast network 126. A subset of the downstream proxy servers 104, 204 interact with the upstream proxy server 102 by the TCP/IP internetwork 124. In some cases the upstream internetwork and the downstream internetwork are actually a single, integrated internetwork. Downstream proxy servers 104, 112, 204 are of one of three types which are referred to as follows:

Reporting downstream proxy servers 104 interact with the upstream proxy server 102 exclusively to satisfy a cache miss. Reporting downstream proxy servers 104 also report cache hits to the upstream proxy server 102. The reporting downstream proxy server 104 is the preferred type of downstream proxy server when the upstream internetwork 124 naturally routes all traffic from the downstream proxy server 104 through a node near the upstream proxy server 102. This is the case when the downstream proxy server 104 is connected to the Internet via a typical, star topology, two-way VSAT network.

Non-reporting downstream proxy servers 112 interact with the addressed web server 110 to satisfy a cache miss. This interaction with the web server 110 may take place either directly with the web server or by means of an upstream proxy server (not shown in FIG. 5) which is independent of the multicast capable upstream proxy server 102. Non-reporting downstream proxy servers 112 do not report cache hits to the upstream proxy server 102. A non-reporting downstream proxy server 112 is the preferred type of downstream proxy server when reporting downstream proxy servers 204 are operating so as to keep the upstream proxy server's estimation of URL popularity up-to-date and to keep the multicast network filled and when a low-complexity minimal processing and memory resources are desired in a subset of the downstream proxy servers.

Best effort downstream proxy servers 204 request a cache-miss URL from both the upstream proxy server 102 and the addressed web server 110. The request to web server 110 may optionally be taken either directly to the web server 110 or by means of an upstream proxy server (not shown in FIG. 5) which is independent of the multicast capable upstream proxy server 102. The best effort downstream proxy server 204 uses the first

8

complete response from either the upstream proxy server 102 or the web server 110, the best effort downstream proxy server 204 is referred to as "best effort" in that best effort communications mechanisms are used between the downstream 204 and the upstream proxy server 102 both to request URLs and to report cache hits. The best effort downstream proxy server 204 is the preferred type of downstream proxy server when the upstream internetwork 124 does not naturally route all traffic from the downstream proxy server near the upstream proxy server. There are many examples where this is the case including where the upstream internetwork 124 is accessed by the downstream proxy server via a dialup modem connection.

As illustrated in FIG. 5a, the upstream proxy server 102 may include:

1. a processor 502 with RAM memory containing programs and data. As is well known in the art, the processor 502 may be implemented in hardware or software, if in hardware, digitally as discrete or integrated circuits. The processor 502 may also include a single processor or multiple processors, interconnected in parallel and/or serial;
2. a multicast transmit network interface 504 capable of transmitting multicast IP packets via the multicast network 126;
3. a point-to-point network interface 506 capable of sending and receiving TCP/IP packets via the upstream TCP/IP Internetwork 124; and
4. a database 508 accessible by the processor 502 containing the status (and optionally the content) of URLs of interest in the upstream proxy server 102.

As is well known to those skilled in the art, a single network interface, such as an ethernet interface, with the proper system routing is capable of carrying both multicast and point-to-point traffic and as such, an alternative implementation of the upstream proxy server 102 may utilize a single network interface 504/506 to carry both the multicast and point-to-point traffic.

As illustrated in FIG. 5b, the downstream proxy server 104, 112, 204 may include:

1. a processor 602 with RAM memory containing programs and data. As is well known in the art the processor 602 may in actual practice be a computer containing a single or multiple processors operating in parallel;
2. a multicast receive network interface 604 capable of transmitting multicast IP packets via the multicast network 126;
3. a point-to-point network interface 606 capable of sending and receiving TCP/IP packets via the upstream TCP/IP Internetwork 124;
4. a point-to-point network interface 607 capable of sending and receiving TCP/IP packets via the downstream TCP/IP Internetwork 1240, 1240', 1240"; and
5. a database 608 accessible by the processor 602 containing a domain name cache containing entries for the domain names or IP addresses of web-servers 110, 110', 110" popularly accessed by clients of the downstream proxy server 104, 112 and a URL cache containing URLs and associated content which can be provided to clients should they request them.

As is well known to those skilled in the art, a single network interface, such as an ethernet interface, with the proper system routing is capable of carrying both multicast and point-to-point traffic and as such, an alternative implementation of the downstream proxy server 104, 112 may

utilize a single network interface to carry both the multicast and upstream TCP/IP Internetwork and downstream TCP/IP Internetwork traffic. Other alternatives include the use of two network interfaces with one of the network interfaces carrying the traffic of two of the network interfaces enumerated above.

The upstream proxy server 102 determines which URLs to multicast and multicasts the URLs and information summarizing URL status. An HTTP URL begins with the string "http://" followed by either the domain name or the IP address of the web server which can serve the URL. The upstream proxy server 102 multicasts URLs in such a way to facilitate the filtering of URLs by web server domain name or IP address. When the upstream proxy server 102 multicasts a URL, it multicasts the URL, the HTTP response header associated with the URL and expiration information for the URL.

Downstream proxy servers 104, 112, 204 maintain a domain name cache with recently accessed URLs. The cache has a maximum size and when a new item is inserted into a full cache, an older, less frequently accessed domain name must be removed to make room for the new item. The downstream proxy servers 104, 112, 204 maintain the relative popularity of each domain name in the cache where popularity is defined by the frequency of HTTP requests to the site. The downstream proxy servers 104, 112, 204 filter out all multicast URLs (and URL status) except those from the most popular entries in the domain name cache. URLs which pass the filter are candidates for being cached.

When a browser 128 requests a URL from the downstream proxy servers 104, 112, 204 the downstream proxy servers 104, 112, 204 update the popularity of that domain name's cache entry adding a new entry for the URL's domain name if not already present. The downstream proxy server 104, 112 then looks up the URL in the downstream proxy server's URL cache. What happens after this depends on whether the URL is found in the cache, whether the URL has expired and whether the downstream proxy server is a reporting 104, non-reporting 112, or best-effort 204 server.

The downstream proxy server 104, 112, 204 directly returns the URL to the browser when the URL is found in the cache and the URL has not expired. A reporting proxy server 104 or a best effort proxy server 204 saves the URL address for subsequent reporting to the upstream proxy server 102. When found and unexpired, both user response time and network utilization are reduced.

The downstream proxy server 104, 112, 204 performs a GET IF MODIFIED SINCE operation against the upstream proxy server 102 when the URL is found and is expired. The downstream proxy server 104, 112, 204 thus checks to make sure the content is up to date.

When a cache lookup finds the URL in the cache and the URL is expired, the processing that takes place depends on the type of downstream proxy server.

A reporting downstream proxy server 104 performs a GET IF MODIFIED SINCE operation against the upstream proxy server 102 when the URL is found and is expired. The reporting downstream proxy server 104 piggybacks any saved URL addresses on the GET IF MODIFIED SINCE request.

A non-reporting downstream proxy server 112 performs a GET IF MODIFIED SINCE operation against the web server 110 when the URL is found and is expired.

A best effort downstream proxy server 204 performs a GET IF MODIFIED SINCE operation against both the upstream proxy server 102 and against the web server 110 in parallel when the URL is found and is expired.

Network utilization and response time are reduced by the present invention in the case of a downstream proxy cache hit of an expired URL provided the GET IF MODIFIED SINCE transaction indicates that the URL has not changed. This is because the actual URL content need not traverse the upstream internetwork 124.

When the URL is not found in the downstream proxy server's cache, the processing that takes place depends on the type of downstream proxy server.

When the cache lookup fails, a reporting downstream proxy server 104 relays the web browser's 128 GET or GET IF MODIFIED SINCE transaction to the upstream proxy server 102 piggybacking any unreported saved URL addresses. As will be discussed later, response time is often reduced even for this case as an HTTP transaction is performed across the proxy-to-proxy link is typically faster than a browser to web server HTTP transaction.

When the cache lookup fails, a non-reporting downstream proxy server 112 relays the web browser's 128 GET or GET IF MODIFIED SINCE transaction to the web server.

When the cache lookup fails, a best effort downstream proxy server 204 relays the web browser's 128 GET or GET IF MODIFIED SINCE transaction to both the upstream proxy server 102 and to the web server. As will be discussed later, response time may be reduced even for this case if the upstream proxy server responds to this transaction and the response arrives sooner than the web server's response.

FIGS. 5c, 5d, and 5e illustrate the processing flow for performing cache lookup for the reporting downstream proxy server 104, non-reporting downstream proxy server 112, and best-effort downstream proxy server 204, respectively.

The upstream proxy server 102 keeps a database of URLs. In some implementations the upstream proxy server 102 is a caching server. When the upstream proxy server 102 is a caching server the URL database may either be integrated with the cache or operate independently of the cache. When the upstream proxy server 102 receives a request for a URL, in some cases it produces a full HTTP response, either from its cache or by interacting with the web server 110 or interacting with a yet further upstream proxy server (not shown in FIG. 5). The upstream proxy server 102 then looks up the URL in its database, updates its entry (or creates the entry if one does not already exist), and determines, based on various criteria discussed later, whether to respond at all and whether to multicast the response. The upstream proxy server 102 returns a point-to-point HTTP response to the reporting downstream proxy server 104 regardless of whether a multicast response is being sent. When a multicast response is being sent, the point-to-point HTTP response signals the reporting downstream proxy server 104 to receive the response via multicast. The upstream proxy server 102 only returns a point-to-point response to a best effort downstream proxy server 204 when the response indicates that the URL is not expired and not modified. Responses containing URL content, when sent to a best effort downstream proxy server 204, are sent only via multicast.

The downstream proxy servers 104, 112, 204 use their domain name caches to efficiently filter and process URLs which have been multicast. The downstream proxy servers 104, 112, 204 discard multicast URLs for domain names not present in the domain name cache. The mechanism for multicasting URLs and for discarding URLs based on domain name is optimized, as will be described in detail later, to reduce the processing required by the downstream

11

proxy servers 104, 112, 204. The downstream proxy servers 104, 112, 204 receive and process multicast URLs for a subset of the domain names in the cache. This subset includes the domain names for which the downstream proxy has an outstanding request to the upstream proxy server 102 and the domain names which the caching policy determines as being most likely to have URLs which will be worth storing in the cache.

A reporting or best effort downstream proxy's 104, 204 domain name cache is organized so that when the proxy server 104, 204 has an HTTP request outstanding to the upstream proxy server 102 that the domain name (or IP address) from the requested URL address is locked in the domain name cache. This ensures that when a response is multicast, the response passes the downstream proxy server's 104, 204 filter and will be processed by the downstream proxy server 104, 204.

When the downstream proxy server 104, 112, 204 receives a multicast URL or URL status update, it submits the URL to its URL caching policy. The caching policy decides whether to store the multicast URL, delete a previously stored URL or expire a previously stored URL. In this way, the downstream proxy server 104, 112, 204 builds up its URL cache with URLs which may be accessed at a later time.

In many systems, such as two-way star-network VSAT systems, both multicast responses and point-to-point responses are carried on a single outbound channel. In such systems, multicasting a response has the benefit of potentially preloading the cache of many receivers while taking no more outbound bandwidth than a point-to-point response. Preloading a cache with a URL reduces network utilization and response time when a successful lookup for that URL occurs at a later time. Overall, the multicasting of responses has the twin benefits of reducing network utilization and response time.

5. PROXY TO PROXY PROTOCOL

5.1 INTRODUCTION

The World Wide Web's use of HTML and the HTTP 1.0 protocol, the version currently in use by almost all browsers, is both inefficient and has very slow response time when operating over satellite networks. This is because HTTP requires a separate TCP connection for each transaction. Multiplying the inefficiency is HTML's mechanism for creating frames and embedding images which requires a separate HTTP transaction for every frame and URL. This particularly affects VSAT networks which have a relatively long round trip delay (1.5 sec) and are cost-sensitive to the number of inbound packets.

FIG. 6 illustrates the packets which traverse the satellite link for a single HTTP transaction and the typical response time. FIG. 7 illustrates the packets which traverse the satellite link (or other network medium) for a single HTTP transaction in the present invention when a reporting downstream proxy server 104 performs the transaction against the upstream proxy server 102. Table 1 illustrates the cumulative effect of this on a HTML page, like www.cnn.com, containing 30 URLs in terms of the total number of inbound packets and the total delay for accessing such a web page. Table 1 also shows the beneficial effects of acknowledgment reduction as described in U.S. Pat. No. 5,985,725 to Dillon entitled "Method and Apparatus for Requesting and Retrieving Information for a Source Computer Using Terrestrial and Satellite Interface" issued Nov. 30, 1999, without which the packet count would be much higher.

12

As can be seen in Table 1, the response time for a 30 URL web page goes from 16.5 seconds without the present invention to 7.5 seconds with the present invention. This is better than a 2 to 1 reduction in response time. As also can be seen from Table 1, the number of inbound packets per 30 URL web page goes from 121 to 30, a better than 4 to 1 reduction.

TABLE 1

Response Time And Inbound Packets For A 30 URL Web Page Over VSAT			
	Inbound Packets	Response Time (sec)	Description
Time For Individual Operation With HTTP 1.0			
A	1	1.5	Domain Name Lookup
B	4	3	One HTTP Get (assuming no web server delay)
Time For An Entire Web Page With HTTP 1.0 (assume 8 browser connections)			
A + B	5	4.5	The HTML URL (first)
B	32	3	First 8 embedded images
B	32	3	Second 8 embedded images
B	32	3	Third 8 embedded images
B	20	3	Last 5 embedded images
	121	16.5	Total For 30 URL Web Page
Time For Individual Operation With Present Invention			
C	0	0	Domain Name Lookup (no satellite round trip, performed by upstream proxy)
D	1	1.5	One HTTP Get (no connection establishment)
E	2	1.5	Last HTTP Get (ack for last data)
Time For An Entire Web Page With Present Invention			
C + D	1	1.5	The HTML URL (first)
D	8	1.5	First 8 embedded images
D	8	1.5	Second 8 embedded images
D	8	1.5	Third 8 embedded images
E	4	1.5	Last 5 embedded images
	29	7.5	Total For 30 URL Web Page

With HTTP 1.1, the proposed enhancement to HTTP 1.0, which improves the response time and networking efficiency of HTTP 1.0, the present invention provides even greater improvements. The present invention provides better compression than HTTP 1.1 and, unlike HTTP 1.1, does not allow a single slow or stalled HTTP request to slow down other requests.

5.2 PROXY-TO-PROXY (P2P) PROTOCOL OVERVIEW AND BENEFITS

The present invention replaces the HTTP protocol often used between upstream 102 and reporting downstream 104 proxy servers with a protocol optimized for this role referred to as the Proxy-To-Proxy (P2P) protocol. The P2P protocol carries HTTP transactions between the downstream 104 and upstream 102 proxy servers.

The P2P protocol carries HTTP 1.0 and 1.1 request and response headers and content where the request and response headers include extensions to support multicast cache pre-loading.

Apart from these multicast header extensions, the P2P protocol improves over HTTP transport in the following ways:

Transaction Multiplexing—improves over separate connection for each transaction (HTTP 1.0) and pipelining

13

(HTTP 1.1) by preventing a single stalled request from stalling other requests. This is particularly beneficial when the downstream proxy server 104 is supporting simultaneous requests from multiple browsers 128, 128', 128".

Homogenized Content Compression—improves over HTTP 1.1 content compression by intelligently compressing HTTP request and response headers and by allowing compression streams for common data to extend over multiple URLs. This increases the overall compression ratio. HTTP 1.1 does not compress request and response headers. This is particularly important when the inbound channel is a shared wireless medium such as a VSAT inroute or some other wireless medium. It effectively allows many more subscribers to share the available inbound bandwidth.

Request Batching—batches HTTP requests which arrive at nearly the same time so that the requests get sent over the satellite in a single TCP segment, thereby reducing the number of inbound packets.

5.3 TRANSACTION MULTIPLEXING

The P2P protocol rides on top of a general purpose protocol, the TCP Transaction Multiplexing Protocol (TTMP). TTMP allows multiple transactions, in this case HTTP transactions, to be multiplexed onto one TCP connection.

The downstream proxy server 104 initiates and maintains a TCP connection to the upstream proxy server 102 as needed to carry HTTP transactions. The TCP connection could be set up and kept connected as long as the downstream proxy server 104 is running and connected to the downstream internetwork 124. It could also be set up when the first transaction is required and torn down after the connection has been idle for some period.

An HTTP transaction begins with a request header, optionally followed by request content which is sent from the downstream proxy server 104 to the upstream proxy server 102. This is referred to as the transaction request. An HTTP transaction concludes with a response header, optionally followed by response content. This is referred to as the transaction response.

The downstream proxy server 104 maintains a transaction ID sequence number which it increments with each transaction. The downstream proxy server 104 breaks the transaction request into one or more blocks, creates a TTMP header for each block, and sends the blocks with a TTMP header to the upstream proxy server 102. The upstream proxy server 102 similarly breaks a transaction response into blocks and sends the blocks with a TTMP header to the downstream proxy server 104. The TTMP header contains the information necessary for the upstream proxy server 102 to reassemble a complete transaction command and to return the matching transaction response. The TTMP header contains:

The transaction ID—the transaction sequence number must rollover less frequently than the maximum number of supported outstanding transactions.

Block Length—allows a proxy server 102, 104, 112, 204 to determine the beginning and ending of each block. As is well known by those skilled in the art, byte stuffing and other techniques can be used, rather than length fields, to identify the beginning and ending of blocks of data.

Last Indication—allows the proxy server 102, 104, 112, 204 to determine when the end of a transaction response has been received.

14

Abort Indication—allows the proxy server 102, 104, 112, 204 to abort a transaction when the transaction request or response cannot be completed.

Compression Information—defines how to decompress the block as explained in more detail in Section 3.4 below.

By breaking transaction requests into blocks and allowing the blocks from different transactions to be interleaved, the P2P protocol of the present invention benefits from allowing a single TCP connection to simultaneously carry multiple HTTP requests without allowing a single stalled (partially received) transaction request or response to block other transactions. The P2P protocol also allows transaction response information to be relayed back to the downstream proxy server 104 in the order it is provided by the various web servers 110, again preventing a stalled or slow web server from delaying URLs from other web servers 110.

The use of a single HTTP connection, rather than the multiple connections used with HTTP 1.0 and optionally with HTTP 1.1 reduces the number of TCP acknowledgements sent over the inbound medium. Reduction in the number of TCP acknowledgements significantly reduces the use of inbound networking resources which, as said earlier, is very important when the inbound is a shared medium such as a VSAT or other wireless medium. This reduction of acknowledgements is more significant when techniques, such as those described in U.S. Pat. No. 5,985,725 to Dillon entitled "Method and Apparatus for Requesting and Retrieving Information for a Source Computer Using Terrestrial and Satellite Interface" issued Nov. 30, 1999, minimize the number of TCP acknowledgements per second per TCP connection.

For example, the Hughes Network Systems DirecPC™ Enterprise Edition networking product reduces the number of TCP acknowledgements per connection sent over the satellite link to two per second regardless of the amount of traffic flowing on that connection. Without the present invention, a web browser 110 might utilize eight HTTP 1.0 connections in parallel across the satellite link. With the present invention, only a single connection is run. As a result, the present invention reduces the inroute acknowledgement traffic eight-fold. If multiple web browsers 110 are operating in parallel, this reduction in inbound acknowledgement traffic is further increased.

5.4 HOMOGENIZED CONTENT COMPRESSION

HTTP 1.1 defines a single algorithm for compressing URL content and each URL's content is individually compressed. The P2P protocol achieves a higher compression ratio than HTTP 1.1 as follows:

Does not restart a compression algorithm with each data item being compressed.

Uses algorithms optimized to the type of data being compressed.

Compresses HTTP request and response headers.

5.4.1 Introduction to Lossless Compression

Lossless compression algorithms can in general be classified into two broad types: statistics-based and dictionary-based. Statistics-based compression algorithms exploit the probability distribution of the data to encode the data efficiently. Two well-known algorithms of this type are Huffman coding and arithmetic coding. The process of statistics-based coding can be split into two parts: a modeler that estimates the probability distribution of data, and an encoder that uses the probability distribution to compress the data. According to information theory, it is possible to

15

construct an optimal code that asymptotically comes arbitrarily close to the entropy rate of the data. Huffman coding can achieve asymptotic optimality by blocking symbols into large groups, however this is computationally infeasible. Moreover, Huffman coding is not suitable for handling adaptive statistics of data. On the other hand, arithmetic coding overcomes these drawbacks of Huffman coding and achieves asymptotic optimality without sacrificing computational simplicity. The essential drawbacks of statistics-based coding are their slower speed (compared to dictionary-based algorithms), and the inaccuracies in statistical modeling of data. Regarding the latter issue, accurate modeling of data requires sophisticated statistical techniques, which in turn require large amount of training data, which is often unavailable.

On the other hand, dictionary-based compression algorithms achieve compression by replacing a string of symbols with indices from a dictionary. The dictionary is a list of strings that are expected to occur frequently in the data. Such a dictionary could either be a static pre-defined dictionary, or an adaptive dictionary that is built and updated as data is compressed. Dictionary-based compression algorithms usually require less computational resources than statistics-based techniques. Lempel-Ziv (LZ) type compression is a general class of adaptive dictionary-based lossless data compression algorithms. LZ-type compression algorithms are based upon two distinct type of approaches—LZ77 and LZ78. An LZ77-type algorithm adaptively builds a dictionary only at the transmitter end of the lossless connection. Compression is achieved by sending a pointer along with size of match of matching phrases occurring in the already compressed portion of the data stream. The receiver end of the connection does not need to maintain a dictionary, thereby minimizing memory requirements, and can decode the compressed data very quickly. LZSS is a commonly used lossless data compression algorithm based on LZ77.

Commonly used LZ78-type lossless data compression algorithms, for example LZW and the algorithm used in the ISO V.42bis specification, use a dictionary (or other data structure) which is built up on both ends of a lossless connection, such as a TCP connection, as data is transferred across the link. Compression is achieved by sending a reference into the dictionary in place of an uncompressed string of bytes. The references are constructed to be smaller than the original string, but sufficient to restore the original string together with the dictionary. These algorithms automatically tune the dictionary as data is transferred so that the dictionary is well prepared to provide high compression should data similar to earlier previously transferred data be submitted for compression.

The term compression stream refers to a compressor and decompressor each with their own dictionary at opposite ends of a lossless connection. For LZ78-type algorithms, encoding is faster than LZ77-type algorithms, however the decoders are slower and have considerably higher memory requirements.

These algorithms are much more efficient when they are processing relatively large amounts of similar data. For example, one HTML page is very similar to subsequent HTML pages, especially when the pages come from the same web server 110. Maximum compression is not obtained immediately after a dictionary is initialized, as it has not been tuned to compress the data at hand.

5.4.2 P2P Use of Lossless Data Compression

The data passed across the P2P protocol can be categorized into the following groups of data:

1. HTTP Request And Response Headers
2. HTML—sent from the upstream proxy 102 to the downstream proxy 104, 112.

16

3. Precompressed data—entity bodies (URL content) such as JPEG and GIF images which are known to be precompressed and do not benefit from further compression attempts.

4. Other—other entity body (URL content) data.

The P2P protocol of the present invention maintains a separate compression stream in each direction for each category of data. This ensures, for example, that downstream HTML data is sent through a compression stream whose dictionary is well tuned for processing HTML providing a higher compression ratio than could be expected from individually compressing each HTML page.

The P2P protocol of the present invention uses compression algorithms for each category of data which are efficient for their category of data. For example, P2P uses:

1. HTTP Requests and Response Headers each are fully text, with plenty of standard keywords. This motivates the use of a dictionary-based algorithm whose dictionary is constructed using the frequently occurring standard keywords. Such a scheme is further improved by combining the static dictionary approach with the power of adaptive dictionary approach of LZ-type algorithm. The static dictionary compresses the standard phrases, while LZ compresses those phrases that are not standard but repeat in the data.
2. HTML data is fully text and warrants the use of a data compression algorithm optimized for text. LZW works well with text and may be used for HTML data when using a more highly optimized algorithm is not convenient.
3. No compression is used for precompressed data to avoid wasting CPU attempting to compress data, which cannot be further compressed.
4. Other data is compressed with a general purpose compression algorithm such as LZW.

As is well known to those skilled in the art, an HTTP request includes an HTTP request header optionally followed by a message body. An HTTP request header further includes a series of one-line, ASCII strings with each string referred to as an HTTP request header field. The end of an HTTP request header is delimited by an empty line. FIG. 8 illustrates a typical HTTP request header.

As is also well known to those in the skilled in the art, an HTTP response includes an HTTP response header optionally followed by a message body. An HTTP response header further includes a series of one-line, ASCII strings with each string referred to as an HTTP response header field. The end of an HTTP response header is delimited by an empty line. FIG. 9 illustrates a typically HTTP response header. The first line of an HTTP response includes the status code, a 3 digit decimal number, which summarizes the type of response being provided. In the example in FIG. 9, a 200 response code is being provided which means that the response was "OK" and that the message body contains the requested URL.

5.5 HTTP 1.1 EXTENSIONS TO SUPPORT MULTICAST CACHE PRELOADING

The present invention includes an addition to a HTTP request used to report cache hit usage reporting and an addition to an HTTP response to direct a reporting downstream proxy server 104 to expect the HTTP response via multicast rather than via P2P. The use of the additions will be explained in detail below.

17

5.5.1 HITREP HTTP Request Extension

By adding a HITREP header field to an HTTP request a reporting downstream proxy 104 or a best effort downstream proxy server 204 may report to the upstream proxy 102 cache hits which occurred on unexpired URLs. A HITREP attribute is formatted as follows:

HITREP=comma separated list of URLs.

The upstream proxy 102 removes this attribute prior to forwarding the HTTP request to the web server 110. It uses the information to update its hit database entries of the contained URLs in a way which increases the likelihood that updates to these URLs will be multicast.

5.5.2 Mcast Status Code

The upstream proxy 102 directs a reporting downstream proxy server 104 to expect the URL to be sent via multicast by returning a Mcast Status Code. In one embodiment, the Mcast Status Code has a value of 360.

5.6 Best Effort Downstream Proxy Server To Upstream Proxy Server Transactions

The best effort downstream proxy server 204, for efficiency, uses the User Datagram Protocol (UDP) to transmit HTTP GET and GET IF MODIFIED SINCE requests to the upstream proxy server 102. This is done by placing the HTTP request header into the UDP payload. In order to piggyback cache hit reports on an HTTP request, the HTTP request header may contain a HITREP field as described earlier.

The use of UDP is very efficient as the overhead of establishing, maintaining and clearing TCP connections is not incurred. It is "best effort" in that lost UDP packets are not recovered. Even with lost packets, the upstream proxy server 102 obtains a very representative view of the URLs of interest to the best effort downstream proxy servers.

6. MULTICAST PROXY TO PROXY PROTOCOL

6.1 URL TRANSPORT

The Advanced Television Enhancement Forum (ATVEF) has published their 1.0 specification containing a description of the Unidirectional Hypertext Transport Protocol (UHTTP). UHTTP defines a method for multicasting URLs. The present invention uses UHTTP with extensions to transport URLs through the multicast network 126 and uses special multicast addressing to minimize the CPU time expended discarding data not of interest.

6.2 MULTICAST ADDRESSING

The multicast system of the present invention is improved to support a single web browser 128 or a small number of browsers 128. When a small number of users (10 or less) is involved, the history of previously visited sites is a strong predictor of future accesses. The present invention leverages this insight to dramatically reduce the processing required to filter multicast URLs. Large scale multicast proxy systems do not leverage this insight as it does not apply significantly when many users are accessing a proxy.

The upstream proxy server 102, when deciding whether to multicast a URL, also classifies the URL as being either of general or specific appeal. General URLs are sufficiently popular to be multicast to all the downstream proxies 104, 112, 204 regardless of the downstream proxy's previous history of sites visited. An example, where this might be applicable would be NASA's Jet Propulsion Laboratory's when the comet hit Jupiter. Many users which had never visited JPL's website would go to that site.

18

The upstream proxy server 102 multicasts general URLs on a single IP multicast address.

To minimize processing of specific URLs, the upstream proxy server 102 spreads the transmission of specific URLs over a large group of multicast addresses. In a preferred embodiment, the upstream proxy server 102 takes the domain of a specific URL and performs a hash function (or any other technique known to one of ordinary skill in the art) to select the multicast address on which the URL is to be multicast. Alternatively, in another embodiment, the upstream proxy server 102 takes the IP address corresponding to the source of the URL and performs a hash function (or any other technique known to one of ordinary skill in the art) on the IP address to select the multicast address on which the URL is to be multicast. The use of hash functions is well understood within the computer science community and is introduced in "Data Structures and Algorithms in C++," Adam Drozdek, PWS Publishing Co., Sections 10-10.1, 1996. The downstream proxy servers 104, 112, 204 utilize the same hash function to determine, from their domain name cache, the set of multicast addresses to open. This mechanism allows a downstream proxy server 104, 112, 204 to use the destination multicast IP address to filter out most of the specific URLs. For downstream proxy servers 104, 112, 204 that have hardware multicast address filtering this effectively eliminates almost all the CPU time spent on filtering specific URLs. Even downstream proxy servers 104, 112, 204 without hardware address filtering can more efficiently filter URLs based on destination address rather than digging into a packet and filtering based the domain name. Filtering based on domain names is based on string comparison operations and is inherently slower than filtering based on unsigned integer comparisons as is the case with hash functions.

Table 2 illustrates how the use of hash function addressing reduces downstream proxy server 104, 112, 204 processing by efficiently limiting the traffic which must be processed.

Hash Function Addressing Filter Effectiveness
50,000 MulticastAddresses Assigned To Carry Specific URLs
100 Domain Names In The Domain Name Cache
1/500 Fraction Of Specific URL Multicast Traffic Passing The Multicast Address Filter

6.3 HTTP 1.1 EXTENSIONS TO SUPPORT MULTICAST PRELOADING

The present invention includes extensions to the HTTP response header to guide a downstream proxy server 104, 112, 204 processing of multicast URLs.

6.3.1 URL Popularity

The upstream proxy server 102 adds a URLPopularity field to an HTTP response. This field identifies the relative popularity of the URL to other URLs which are being multicast. The URL Popularity field holds an 8 digit unsigned hexadecimal number. The field contains the Age-dAccessNumber further discussed below.

6.3.2 Mcast Expiration

The upstream proxy server 102 also adds a McastExpiration header field to an HTTP response. This field contains, like an Expires field, an HTTP-date field. It may also contain 0 which means consider the URL expired. The downstream proxy server 104, 112, 204 uses this field to determine whether to validate its URL cache entry by making a GET IF MODIFIED SINCE request.

7. UPSTREAM PROXY MULTICAST POLICY

It is expected that the upstream proxy server's 102 multicast policy will be improved over time. The implemen-

19

tation of the present invention allows this policy to be enhanced without disrupting the operation of the receiver. The policy described here provides a clear mechanism for reducing overall outbound network utilization.

7.1 URL ADDRESS DATABASE

The upstream proxy server 102, as described earlier, receives from reporting and best effort downstream proxy servers 104, 204 requests for URLs and cache hit reports. The upstream proxy server 102 uses these requests and usage reports to maintain a URL address database. This database contains URL address entries, each of which contains:

URL Address—the URL address itself or a message digest of the address (see the discussion of message digests below).

AgedAccessCounter—a 32-bit unsigned counter which is increased with every request for the URL and with each usage report for the URL and which is reduced to age out stale entries.

ExpirationTime—holds the GMT time when this URL expires.

The upstream proxy server 102 maintains the AgedAccessCount such that it is an indicator of its URL's popularity, that is, frequency of access by downstream proxy servers 104, 112. The upstream proxy server 102, upon receiving a request or a usage report for a URL, looks up the URL in its database, if found, increases its AgedAccessCount, for example, by 1000. The upstream proxy server 102 creates an entry with the AgedAccessCount initialized to a default initial value (e.g. 1000) if the URL was not found. Periodically, (e.g. hourly), the upstream proxy server 102 reduces each database entry's AgedAccessCount by a configurable amount (e.g. 10%).

7.2 MESSAGE DIGESTS

As is well known to a practitioner skilled in the art, a message digest (or digest) is a relatively short (e.g. 64 bits), fixed length string of bits which is a function of a variable length string of bits. This function has the property that the message digest of different variable length strings will almost always have different digests. "Almost always" means, in this case, a very low probability (e.g. 1 in 2^{60} or one in 10^{18}). Some message digest functions also have the useful property in cryptographic systems that it is difficult to create a string whose message digest is identical to the message digest of another string. This property is not required for this invention. Message digests are introduced in "Applied Cryptography" by Bruce Schneier. The present invention utilizes message digests to determine when two URL addresses are identical (by checking whether their digests are identical).

7.3 MULTICAST NETWORK UTILIZATION

The upstream proxy server 102 is configured with a maximum multicast outbound bit rate, for example, 6 Mbits/sec. The upstream proxy server 102 manages its multicast transmissions to not exceed this maximum rate. In the preferred embodiment, the upstream proxy server 102 maintains twenty byte counters, one for each tenth of a second. It moves round robin from one counter to the next every tenth of a second. When it multicasts a packet, it adds the size of the packet to the counter. Thus, the upstream proxy server 102 can calculate the average throughput over the last two seconds. From this average throughput, the upstream

20

proxy server 102 can calculate the overall multicast utilization, that is, the average throughput divided by the maximum multicast outbound bit rate.

The upstream proxy server 102 is also configured with a maximum general multicast outbound bit rate, for example, 1 Mbit/sec. The upstream proxy server 102 manages its multicast transmission of general URLs to not exceed this maximum rate. This is done in a fashion similar to overall multicast transmission, the upstream proxy server 102 can calculate its general multicast utilization.

7.4 HTTP RESPONSE EXPIRATION FIELD

The upstream proxy server 102 receives HTTP responses either from a web server 110 or a yet further upstream proxy server (not shown in FIG. 5). Prior to multicasting a URL, the upstream proxy server 102 must ensure that there is an appropriate expiration field in the HTTP response header.

The policy for calculating the expiration is as follows:

1. If any cookies were present in the HTTP request, the response may be specific to the requesting browser and the upstream proxy server 102 sets the Expiration field to 0, indicating already expired. As is well known to those skilled in the art, a cookie is a data item which is provided by a web server to a browser in an HTTP response and is returned to the web server in subsequent HTTP requests. It is typically used to allow the web server to identify the requests which are coming from a single user. A cookie HTTP request header field is shown in the typical request illustrated in FIG. 8.
2. Otherwise, if the expiration field already exists, the upstream proxy server 102 leaves it untouched.
3. Otherwise, the upstream proxy server 102 sets the expiration field based on MIME type. The upstream proxy server 102 is configured with a table giving the expiration duration for various MIME types and a default expiration for all other MIME types. The upstream proxy server 102 takes the current GMT time and adds the appropriate expiration duration to calculate the expiration time. This allows HTML (which is more likely to change) to expire sooner than images (gif and jpg) which are less likely to change).

7.5 MULTICAST DECISION

The upstream proxy server 102 receives HTTP responses either from a web server 110 or a yet further upstream proxy server (not shown in FIG. 5). The upstream proxy server 102 examines the HTTP response header to determine the cachability of the URL.

If it is uncachable and the request came from a reporting downstream proxy server, the upstream proxy server 102 returns the response to the downstream proxy server 104 by its point-to-point connection. If it is cacheable, the upstream proxy server 102 looks up its URL in the URL address database and determines whether to return a response to the downstream proxy server 104, 204 and how that response is returned. A response must be returned to a reporting downstream proxy server 104. The response may be sent either via multicast or via point-to-point connection. A response, if necessary, is returned to a best effort downstream proxy server 204, via multicast. Multicast responses may either be sent on the general or on a specific multicast address. The preferred embodiment of the present invention may utilize the following algorithm to determine how, if at all, the upstream proxy server returns a response.

1. Determine whether the URL is qualified to be general multicast, multicast on the general address if qualified.

2. If not, determine whether the URL is qualified for specific multicast, multicast on a specific address if qualified.
3. Otherwise, send it point-to-point if the request came from a reporting downstream proxy server. Send no response otherwise.

7.5.1 General Multicast Decision

The general multicast decision is based on whether the URL content is included in the response, the popularity of the URL and the general multicast utilization where as the utilization goes up, the popularity of the URL also must go up for the URL to be qualified to be transmitted.

The URL content is not included in a "not modified" response to a GET IF MODIFIED SINCE request. Such a response is only qualified to be multicast when the corresponding entry in the URL address database is "expired" and the response itself is not expired. A "qualified" response with no URL content is worth multicasting as it may be used to update the expiration time of the corresponding entry in the cache of the downstream proxy servers.

The upstream proxy server 102 is configured with a general multicast decision table. This table contains a set of entries, each entry containing a general multicast utilization threshold and a minimum AgedAccessCount. A URL is qualified for general multicast transmission if there is any entry in the table where the general multicast utilization is less than the general multicast utilization threshold and AgedAccessCount exceeds the minimum AgedAccessCount. To avoid overloading the general multicast maximum bit rate, the table always contains an entry for 100% utilization which requires an infinitely high AgedAccessCount and the table allows no other entries with a utilization of 100% or higher.

7.5.2 Specific Multicast Decision

The specific multicast decision is based on the popularity of the URL and the overall multicast utilization where, as the utilization goes up, the popularity of the URL also must go up for the URL to be qualified to be transmitted.

The upstream proxy server 102 is configured with a specific multicast decision table. This table contains a set of entries, each entry containing a overall multicast utilization threshold and a minimum AgedAccessCount. A URL is qualified for specific multicast transmission if there is any entry in the table where the specific multicast utilization is less than the specific multicast utilization threshold and AgedAccessCount exceeds the minimum AgedAccessCount. To avoid overloading the overall multicast maximum bit rate, the table always contains an entry for 100% utilization which requires an infinitely high AgedAccessCount and the table allows no other entries with a utilization of 100% or higher.

7.5.3 Preferred Site Access To Multicast

It may be desirable to give certain web sites preferred access to the multicast channel. The present invention accommodates this by allowing "preferred" reporting and best effort downstream proxy servers 104, 204 to be configured and to configure the upstream proxy server 102 to preferentially multicast requests from "preferred" reporting and best effort downstream proxy servers 104, 204. The upstream proxy server 102 multicasts all responses to requests coming from a "preferred" downstream proxy server 104, 204, giving the site priority to the multicast bandwidth and queuing the responses until bandwidth is available. A web crawler program, such as Teleport Pro by Tennyson Maxwell (www.tenmax.com) is then programmed to periodically crawl such a preferred web site. This results in the preferred web site's content being periodically mul-

ticast. A preferred downstream proxy server 104, 112 can be configured to either have its responses multicast either as general multicasts (for sites which are very much preferred) or as specific multicasts (for sites which are preferred).

8. DOWNSTREAM PROXY CACHING POLICY

8.1 CACHING POLICY OVERVIEW

It is expected that the downstream proxy server's 104, 112, 204 cache policy will also be improved over time. The implementation of the present invention allows this policy to be enhanced without changing the interface to the upstream proxy server 102. The policy described here provides a clear mechanism for reducing overall outbound network utilization.

The cache policy of the preferred embodiment of the present invention is optimized for small-scale operation where the downstream proxy server 104, 112, 204 is supporting either a single browser 128 or a small number of browsers and where these browsers have their own caches. The policy supplements the benefits of a browser cache with most of the benefits of the large-scale cache while consuming a fraction of a large-scale cache's resources.

The cache policy includes four separate operations:

1. determining which multicast addresses to open;
2. determining what to do with URLs received on those addresses;
3. aging cache entries in a fashion identical to the upstream proxy server's URL address database entry aging; and
4. cache lookup.

8.2 MULTICAST ADDRESS POLICY

8.2.1 Multicast Reception Modes

The multicast receiver for the downstream proxy server 104, 112, 204 operates in one of two modes:

active—the downstream proxy server 104, 112, 204 opens multicast addresses and actively processes the received URLs on those addresses.

inactive—the downstream proxy server 104, 112, 204 disables multicast reception from the upstream proxy server 102. In the inactive state the downstream proxy server 104, 112, 204 minimizes its use of resources by, for example, closing the cache and freeing its RAM memory.

For downstream proxy server 104, 112, 204 operating on a general purpose personal computer, the multicast receiver for the downstream proxy server 104, 112, 204 may be configured to switch between the active and inactive states to minimize the proxy server's interfering with user-directed processing. The downstream proxy server 104, 112, 204 utilizes an activity monitor which monitors user input (key clicks and mouse clicks) to determine when it should reduce resource utilization. The downstream proxy server 104, 112, 204 also monitors for proxy cache lookups to determine when it should go active.

Upon boot up, the multicast receiver is inactive. After a certain amount of time with no user interaction and no proxy cache lookups (e.g. 10 minutes), the downstream proxy server 104, 112, 204 sets the multicast receiver active. The downstream proxy server 104, 112, 204 sets the multicast receiver active immediately upon needing to perform a cache lookup. The downstream proxy server 104, 112, 204 sets the multicast receiver inactive whenever user activity is detected and the cache has not had any lookups for a configurable period of time (e.g. 5 minutes).

23

For downstream proxy servers 104, 112, 204 running on systems with adequate CPU resources to simultaneously handle URL reception and other applications, the user may configure the downstream proxy server 104, 112, 204 to set the multicast receiver to stay active regardless of user activity.

8.2.2 Multicast Address Selection

The downstream proxy server 104, 112, 204 is configured to open a configurable number of multicast addresses, for example, 150 addresses. When the downstream proxy server sets the multicast receiver active, the downstream proxy server 104, 112, 204 always opens the general multicast address and the specific multicast addresses for the web sites for which it has outstanding requests to the upstream proxy server 102. It opens the specific addresses corresponding to the most popular domain names in its domain name cache with the remaining address slots. Reporting downstream proxy servers and best effort downstream proxy servers give priority to the domain names of URLs for which they have outstanding HTTP requests open to the upstream proxy server and close specific addresses as needed to make room for the addresses associated with those URLs. The downstream proxy server 104, 112, 204 thus has access to the multicast of the web sites it is most likely, based on past history, to access.

8.2.3 Multicast URL Reception Processing

A downstream proxy server 104, 112, 204 may receive via multicast either a complete HTTP response with the URL content or "not modified" HTTP response header without URL content with an updated McastExpiration field.

The downstream proxy server 104, 112, 204 examines the URL popularity field of a complete HTTP response and removes URLs from the cache until there is room for the URL just received. The downstream proxy server removes URLs beginning with those with the lowest AgedAccessCounter values. The downstream proxy server 104, 112, 204 discards a received URL when there are insufficient URLs whose AgedAccessCounter fields are lower than the URL popularity field of the URL just received to make room for the URL just received. When storing the URL just received in the cache, the downstream proxy server 104, 112, 204 copies the URL popularity field into the cache entry's AgedAccessCount.

Upon receiving a "not modified" HTTP response header without URL content, the downstream proxy server 104, 112, 204 looks up the corresponding URL in its cache. If found, the downstream proxy server updates the cache entry's AgedAccessCounter value with the URL popularity field and updates the entry's expiration field with the response header's McastExpiration field's value.

After updating the cache, a reporting or best effort downstream proxy server 104, 204 checks whether an HTTP request is outstanding to either the webserver or upstream proxy server for the received URL. If so and the URL is now in the cache, the downstream proxy server responds to the requesting browser with the cache entry. If a "not-modified" URL response was received and a request for the URL is outstanding and there was no cache entry the downstream proxy server 104, 204 returns the "not-modified" HTTP response to the browser.

8.2.4 URL Cache Aging

The downstream proxy server 104, 112, 204 ages URLs the same way the upstream proxy 102 ages URLs.

8.2.5 Cache Lookup

When the downstream proxy server 104, 112, 204 looks up a URL in the cache and the URL has not expired, the downstream proxy server 104, 112, 204 returns the URL

24

from the cache to the browser 128. When the URL has expired, the downstream proxy server 104, 112, 204 sends a GET IF MODIFIED SINCE transaction against the upstream proxy server 102 and/or the web server 110 as is appropriate for the category of proxy server receiving the request.

9. CONCLUSION

As set forth above, the present invention offers many significant innovations over prior satellite systems multicast systems by offering lower response time and lower network utilization while limiting the resources required within the satellite receiver and associated equipment needed to achieve these benefits.

Although several embodiments of the present invention have been described above, there are of course numerous other variations that would be apparent to one of ordinary skill in the art. For example, one or more of the downstream proxy servers 104, 112, 204 could reside with the browser 128, 128', 128" on a single personal computer 122, 122', 122". Additionally, one or more of the downstream proxy servers 104, 112, 204 could reside with the browser 128, 128', 128" on a television set-top box. Further, one or more of the downstream proxy servers 104, 112, 204 residing with the browser 128, 128', 128" on a single personal computer 122, may also have a downstream TCP/IP internetwork connection to other browsers which may or may not be operating on personal computers. Also, one or more of the downstream proxy servers 104, 112, 204, residing with a browser 128, 128', 128" on a television set-top box, may also have a downstream TCP/IP internetwork connection to other browsers which may or may not be operating on personal computers. Also, the multicast network 126 need not be based on geosynchronous satellite technology but could be based on any of a number of other multicast technologies including wireless terrestrial broadcast systems.

The invention being thus described, it will be obvious that the same may be varied in many ways. Such variations are not to be regarded as a departure from the spirit and scope of the invention, and all such modifications as would be obvious to one skilled in the art are intended to be included within the scope of the following claims.

What is claimed is:

1. A communications system comprising:

at least one upstream proxy server; and

at least two reporting downstream proxy servers;

said at least one upstream proxy server capable of multicasting URLs to said at least two reporting downstream proxy servers;

said at least two reporting downstream proxy servers interacting with said at least one upstream proxy server to resolve cache misses;

wherein said at least one upstream proxy server returns at least one resolution to the cache misses via multicast,

where said at least two reporting downstream proxy servers utilize a relative frequency that a source web server of a multicast URL has had items requested by clients of at least one of said at least two reporting downstream proxy servers to determine whether to store a multicast URL.

2. The communication system of claim 1, where said at least one upstream proxy server returns at least one response to the cache misses via point-to-point transmission.

3. The communication system of claim 2, where said at least two reporting downstream proxy servers send cache hit information to said at least one upstream proxy server.

25

4. The communication system of claim 3, where said at least two reporting downstream proxy servers piggyback cache hit information on HTTP request headers being sent to said at least one upstream proxy server.

5. The communication system of claim 4, where said at least one upstream proxy server uses relative frequency accesses of the URL including both cache misses and cache hits as reported to the upstream server to help determine whether content of a cache miss is returned via multicast or point-to-point transmission.

6. The communication system of claim 2, where said at least one upstream proxy server uses popularity, where popularity is based on a relative frequency of access of a URL, to determine whether the URL is returned via multicast or point-to-point transmission.

7. The communication system of claim 5, where said at least one upstream proxy server also uses a loading of the multicast channel in combination with the popularity to determine whether the URL is returned via multicast or point-to-point transmission.

8. The communication system of claim 2, where said at least one upstream proxy server maps a domain name of a source of a URL to a multicast address to determine the multicast address to be used to carry the URL.

9. The communication system of claim 8, wherein said at least one upstream proxy server maps domain names to multicast addresses using of a hash function.

10. The communication system of claim 8, wherein URLs with a relatively high popularity are carried on a multicast address dedicated to carrying URLs of general interest.

11. The communication system of claim 1, wherein additional reporting downstream proxy servers which are not currently interacting with said at least one upstream proxy server filter multicast cache resolutions from said at least one upstream proxy server and store a subset of cacheable items for subsequent retrievals upon request, by a client.

12. The communication system of claim 1, wherein additional non-reporting downstream proxy servers which do not report to said at least one upstream proxy server filter multicast cache resolutions from said at least one upstream proxy server and store a subset of cacheable items for subsequent retrieval, upon request, by a client.

13. A communication system comprising:

at least one multicast capable upstream proxy server; and
at least two best-effort downstream proxy servers;

said at least one multicast capable upstream proxy server capable of multicasting URLs to said at least two best-effort downstream proxy servers where said at least two best-effort downstream proxy servers interact with said at least one upstream proxy server and either a web-server directly or at least one non-multicast capable proxy server to resolve cache misses;

wherein said at least one multicast capable upstream proxy server returns at least one resolution to the cache misses via multicast; and

26

wherein said at least two best-effort downstream proxy servers relay responses from said at least one multicast capable proxy server to a client when the responses arrive prior to a response from the web server or said at least one non-multicast capable upstream proxy server.

14. The communication system of claim 13, where said at least two best-effort downstream proxy servers use a best-effort communication mechanism to send cache miss resolution requests to said at least one multicast capable upstream proxy server.

15. The communication system of claim 13, where said at least two best-effort downstream proxy servers send cache hit information to said at least one multicast capable upstream proxy server.

16. The communication system of claim 15, where said at least two best-effort downstream proxy servers piggyback cache hit information on HTTP request headers sent to said at least one multicast capable upstream proxy server.

17. An upstream proxy server capable of multicasting URLs to at least two reporting downstream proxy servers; said upstream proxy server interacting with said at least two reporting downstream proxy servers to resolve cache misses;

wherein said upstream proxy server returns at least one resolution to the cache misses via multicast,

where said at least one upstream proxy server returns at least one response to the cache misses via point-to-point transmission,

where the upstream proxy server is able to receive cache hit information from at least one downstream proxy server,

where the upstream proxy server uses a relative frequency of cache misses and cache hits to an individual server to determine whether content of a cache miss is returned via multicast or point-to-point transmission.

18. The upstream proxy server of claim 17, where the upstream proxy server is able to receive cache hit information from at least one downstream proxy server piggybacked on an HTTP request from said downstream proxy server.

19. The upstream proxy server of claim 17, where the upstream proxy server uses popularity, where popularity is based on a relative frequency of access of a URL, to determine whether the URL is returned via multicast or point-to-point transmission.

20. The upstream proxy server of claim 19, where the upstream proxy server also uses a loading of the multicast channel in combination with the popularity to determine whether the URL is returned via multicast or point-to-point transmission.

* * * * *



US006128653A

United States Patent [19]
del Val et al.[11] **Patent Number:** **6,128,653**
[45] **Date of Patent:** **Oct. 3, 2000**[54] **METHOD AND APPARATUS FOR
COMMUNICATION MEDIA COMMANDS
AND MEDIA DATA USING THE HTTP
PROTOCOL**[75] Inventors: **David del Val**, Mountain View; **Anders
Edgar Klemets**, Sunnyvale, both of
Calif.[73] Assignee: **Microsoft Corporation**, Redmond,
Wash.[21] Appl. No.: **08/822,156**[22] Filed: **Mar. 17, 1997**[51] Int. Cl.⁷ **G06F 13/14**[52] U.S. Cl. **709/219; 709/203; 455/4.2**[58] Field of Search **345/327-329;
395/200.47-200.49, 200.33; 455/4.2; 348/7,
12, 13; 709/217-219, 203**[56] **References Cited****U.S. PATENT DOCUMENTS**

4,931,950	6/1990	Isle et al.	364/513
5,119,474	6/1992	Beitel et al.	395/154
5,274,758	12/1993	Beitel et al.	395/154
5,341,474	8/1994	Gelman et al.	395/200
5,414,455	5/1995	Hooper et al.	348/7
5,455,910	10/1995	Johnson et al.	395/650
5,533,021	7/1996	Branstad et al.	370/60.1
5,537,408	7/1996	Branstad et al.	370/79
5,623,690	4/1997	Palmer et al.	395/806
5,721,827	2/1998	Logan et al.	395/200.47
5,732,219	3/1998	Blumer et al.	395/200.47
5,754,772	5/1998	Leaf	395/200.33
5,793,966	8/1998	Amstein et al.	395/200.33
5,796,393	8/1998	MacNaughton et al.	345/329

5,796,566 8/1998 Sharma et al. 361/86

FOREIGN PATENT DOCUMENTS

0605115	7/1994	European Pat. Off.
0653884	5/1995	European Pat. Off.
0676898	10/1995	European Pat. Off.
0746158	12/1996	European Pat. Off.

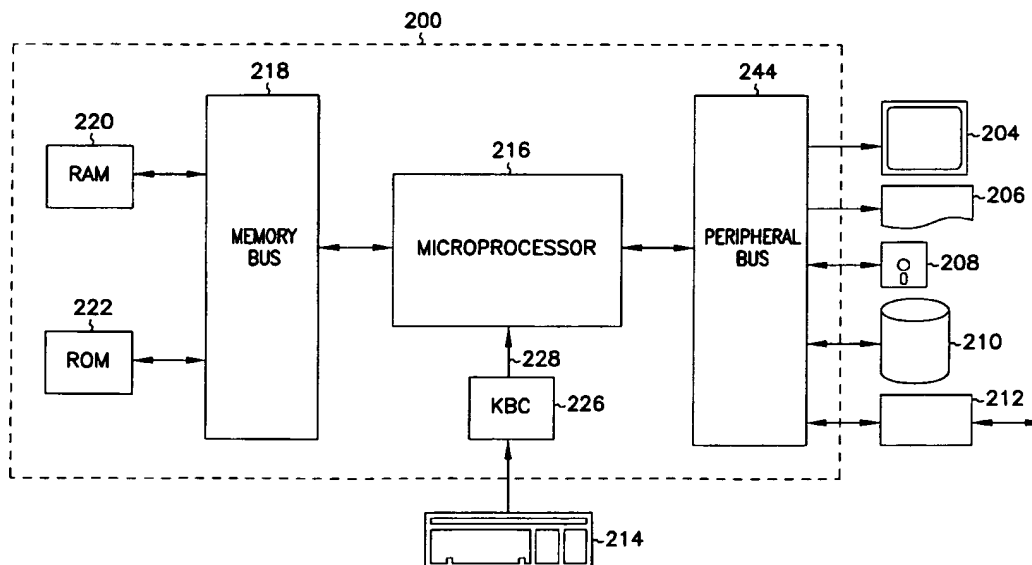
OTHER PUBLICATIONS

Chen, H.J., et al., "A Scalable Video-on-Demand Service for the Provision of VCR-Like Functions", IEEE Proceedings of the International Conference on Multimedia Computing and Systems, Washington, DC, 65-72, (May 15-18, 1995).

Primary Examiner—Victor R. Kostak
Attorney, Agent, or Firm—Schwegman, Lundberg,
Woessner & Kluth, P.A.

[57] **ABSTRACT**

A method for employing a Hypertext Transfer Protocol (HTTP protocol) for transmitting streamed digital media data from a server. The server is configured for coupling to a client computer via a computer network. The method includes receiving at the server from the client an HTTP POST request. The POST request requests a first portion of the digital media data and includes a request header and a request entity-body. The request entity body includes a media command for causing the first portion of the digital media data to be sent from the server to the client. The method further includes sending an HTTP response to the client from the server. The HTTP response includes a response header and a response entity body. The response entity body includes at least a portion of the first portion of the digital media data.

35 Claims, 11 Drawing Sheets

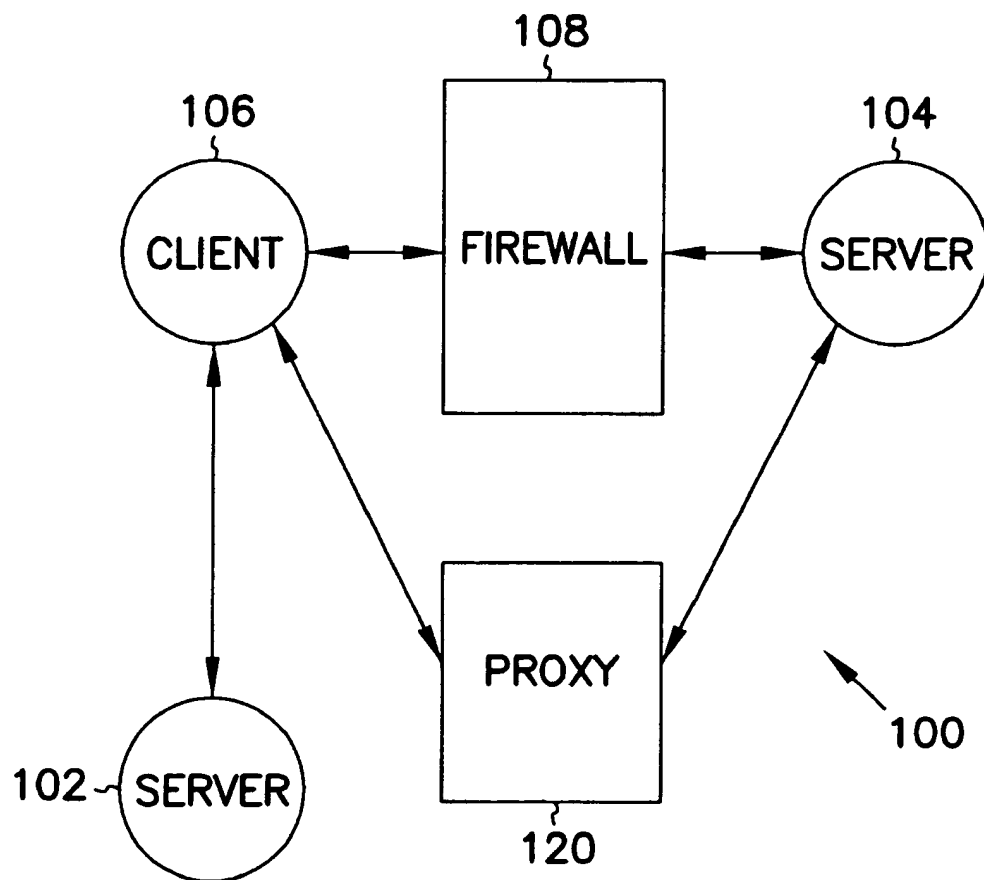


FIG. 1

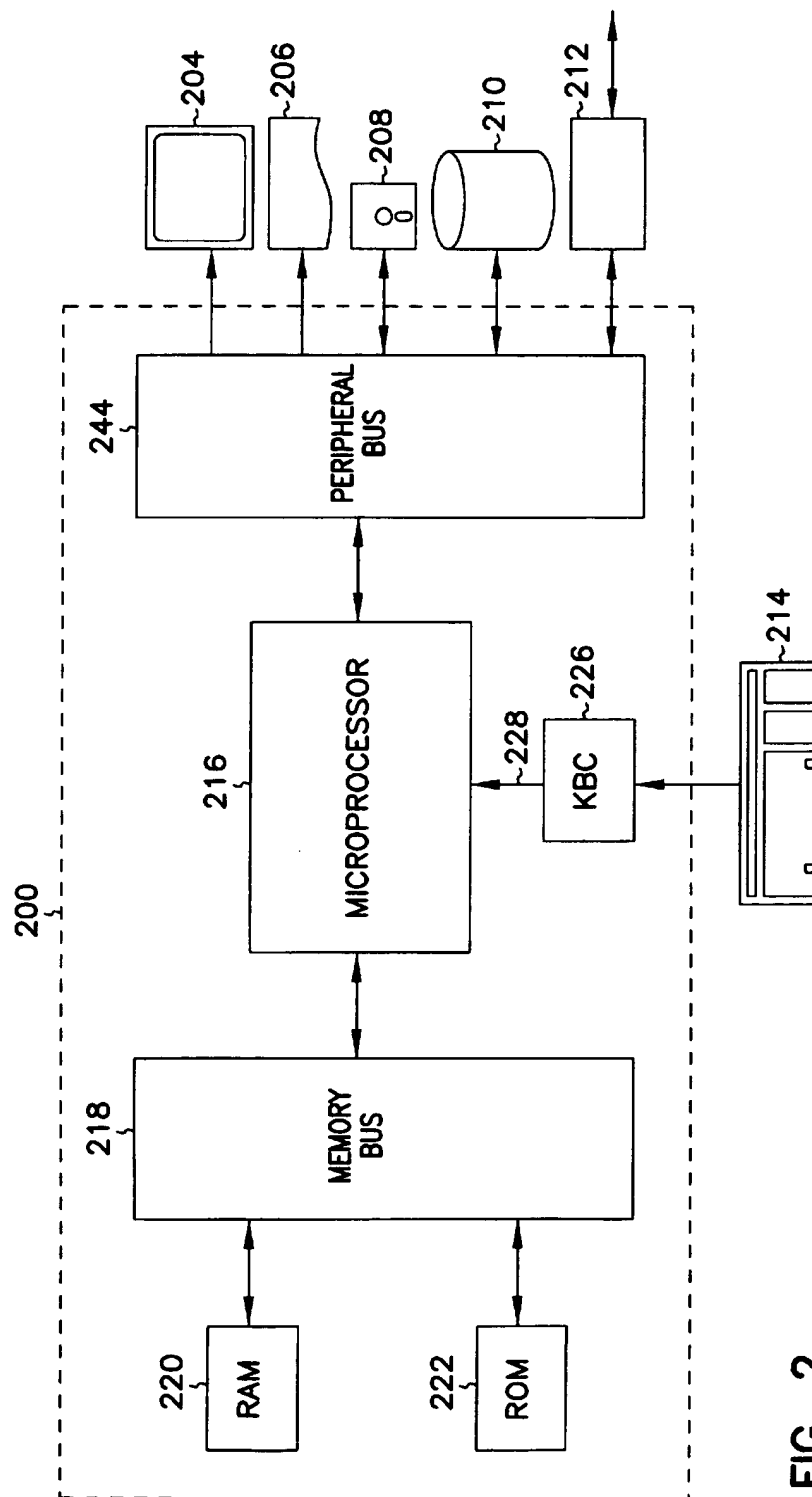
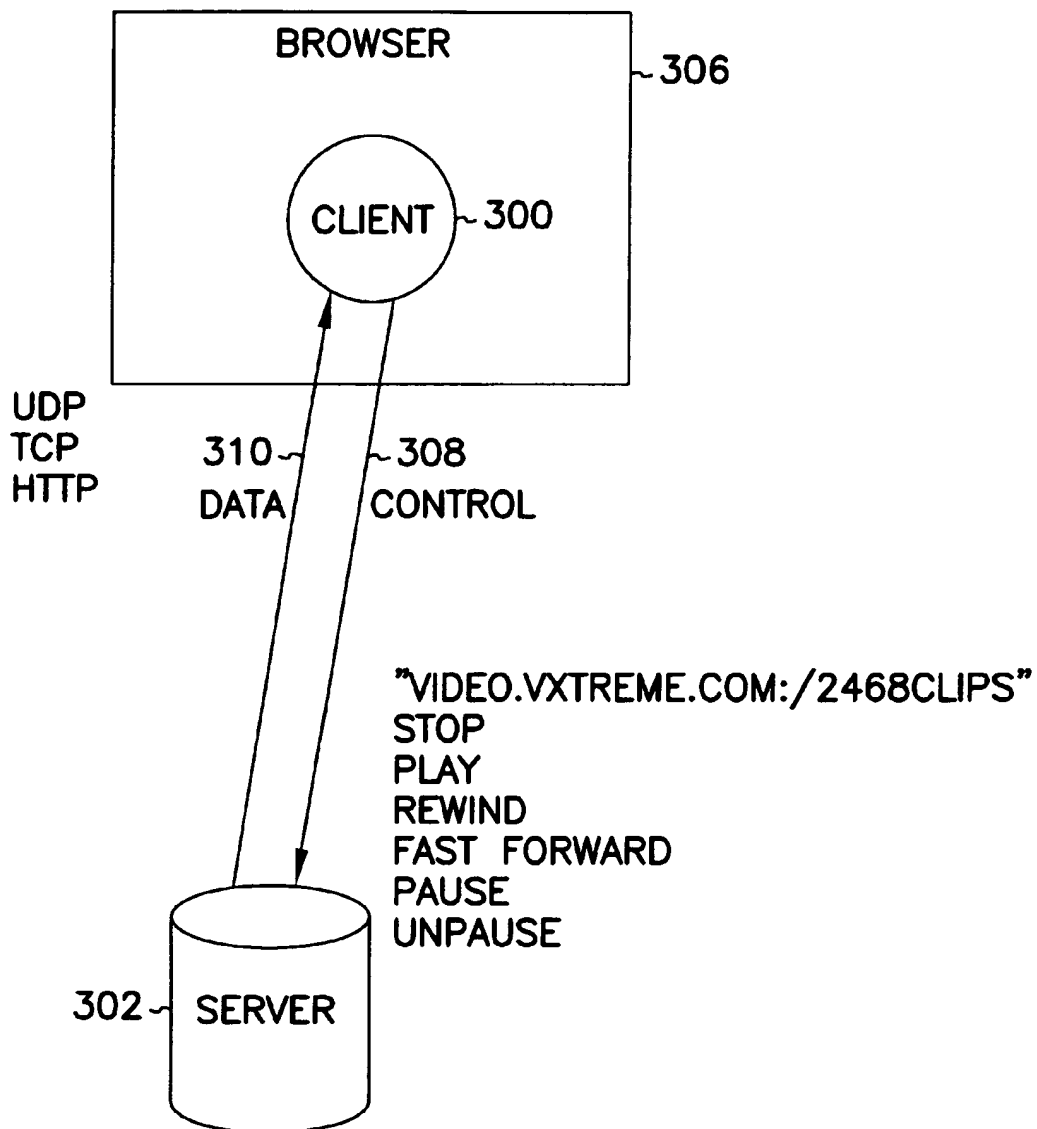
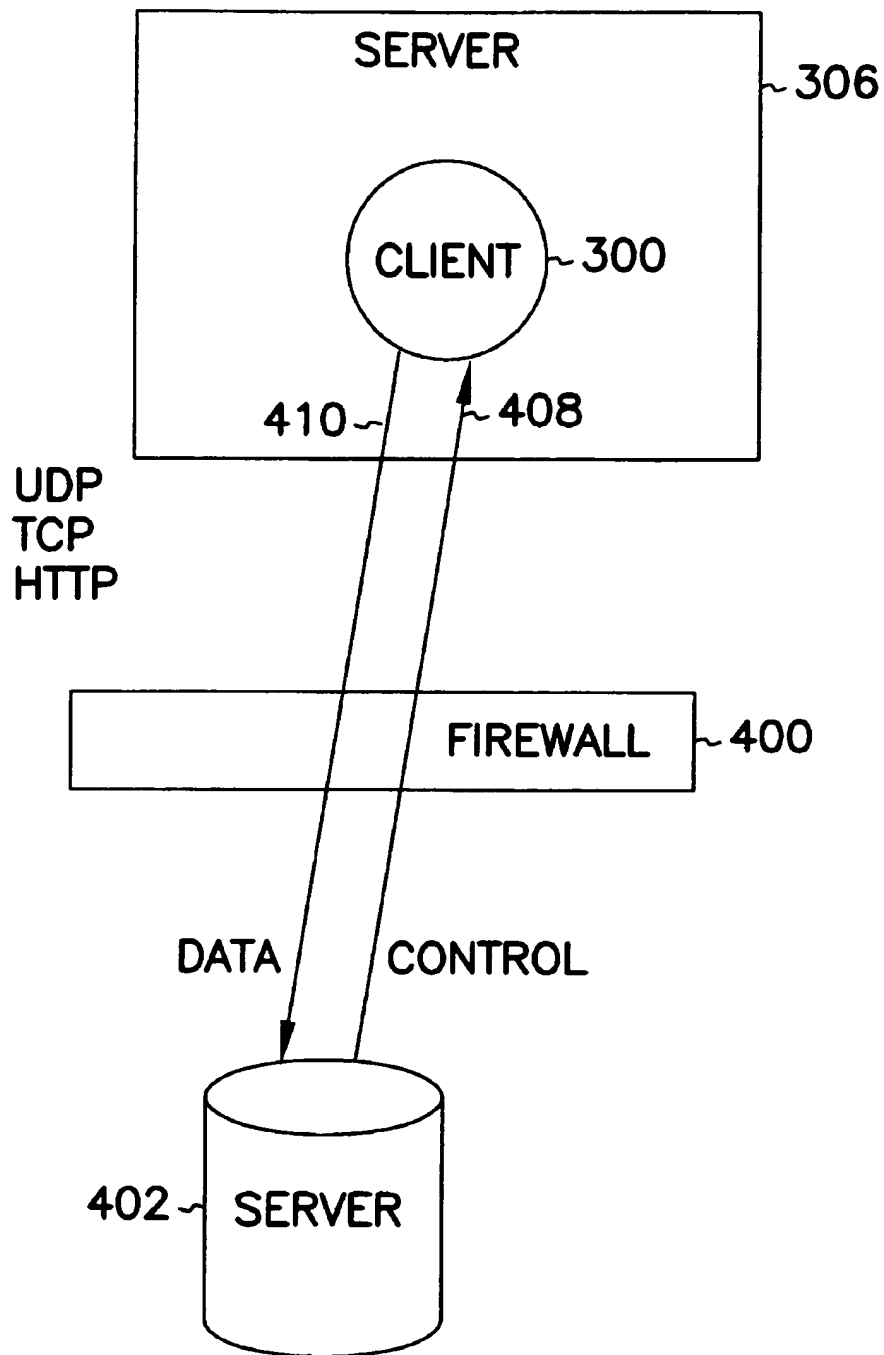


FIG. 2

**FIG. 3**

**FIG. 4**

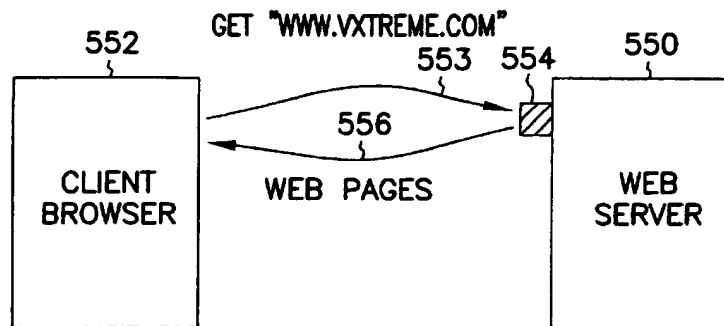


FIG. 5A (PRIOR ART)

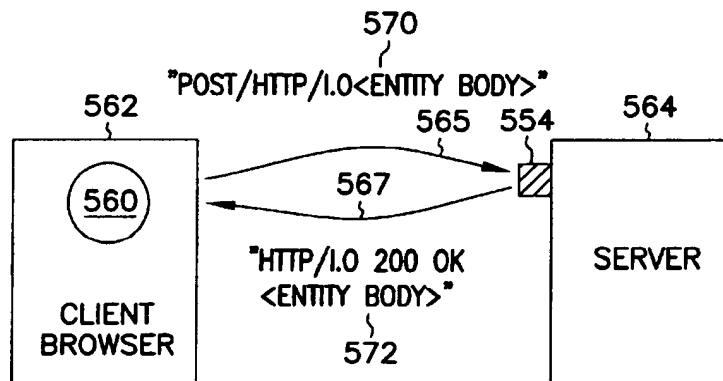


FIG. 5B

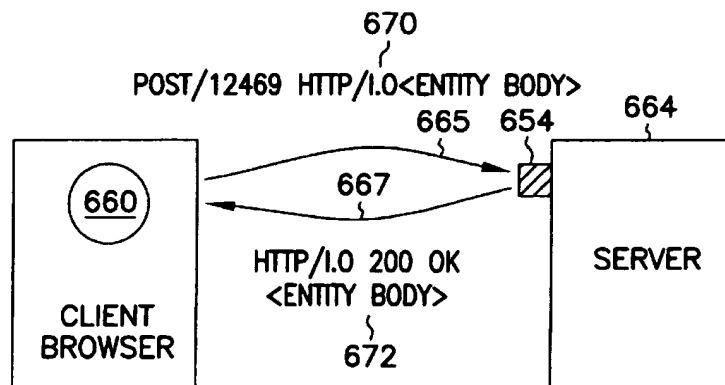


FIG. 5D

<ENTITY BODY>:
AUDIO:29,999
OR
VIDEO : 35,122

672

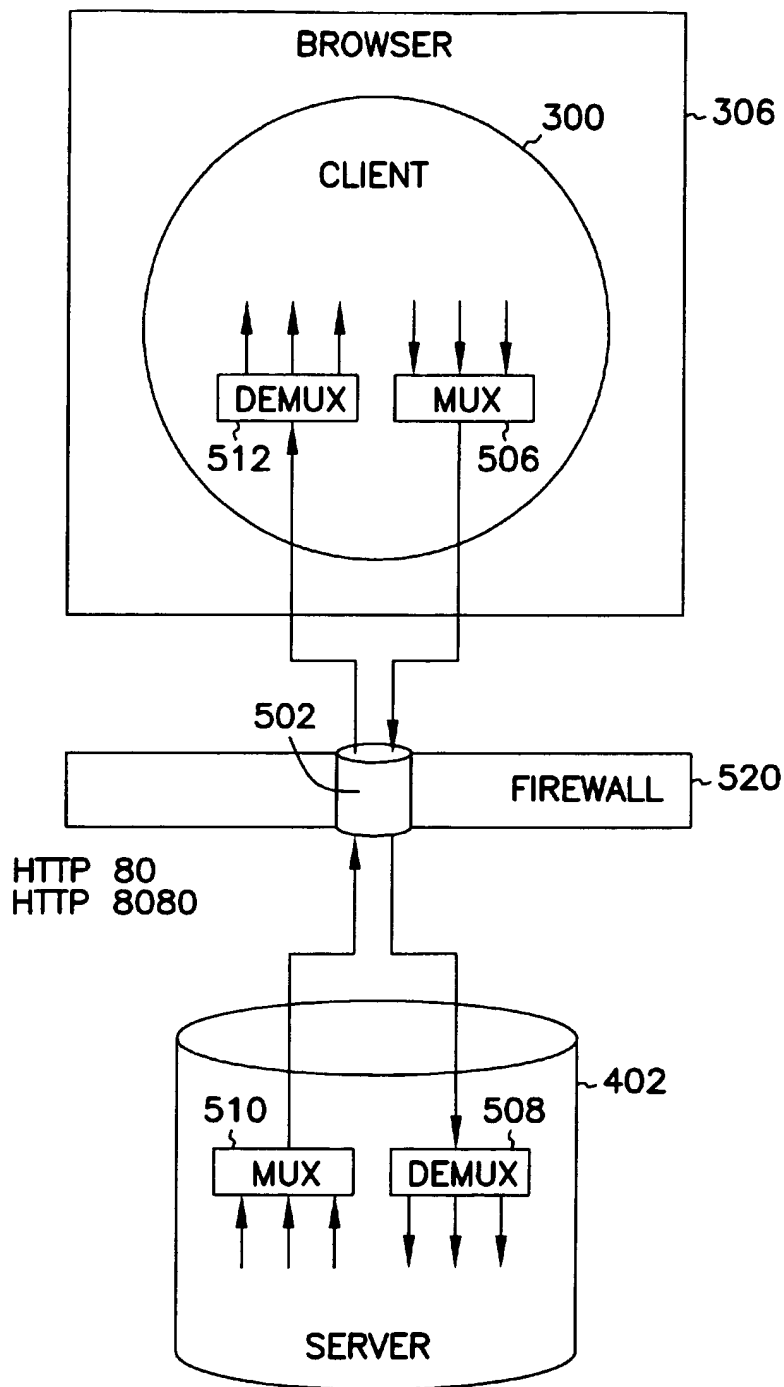
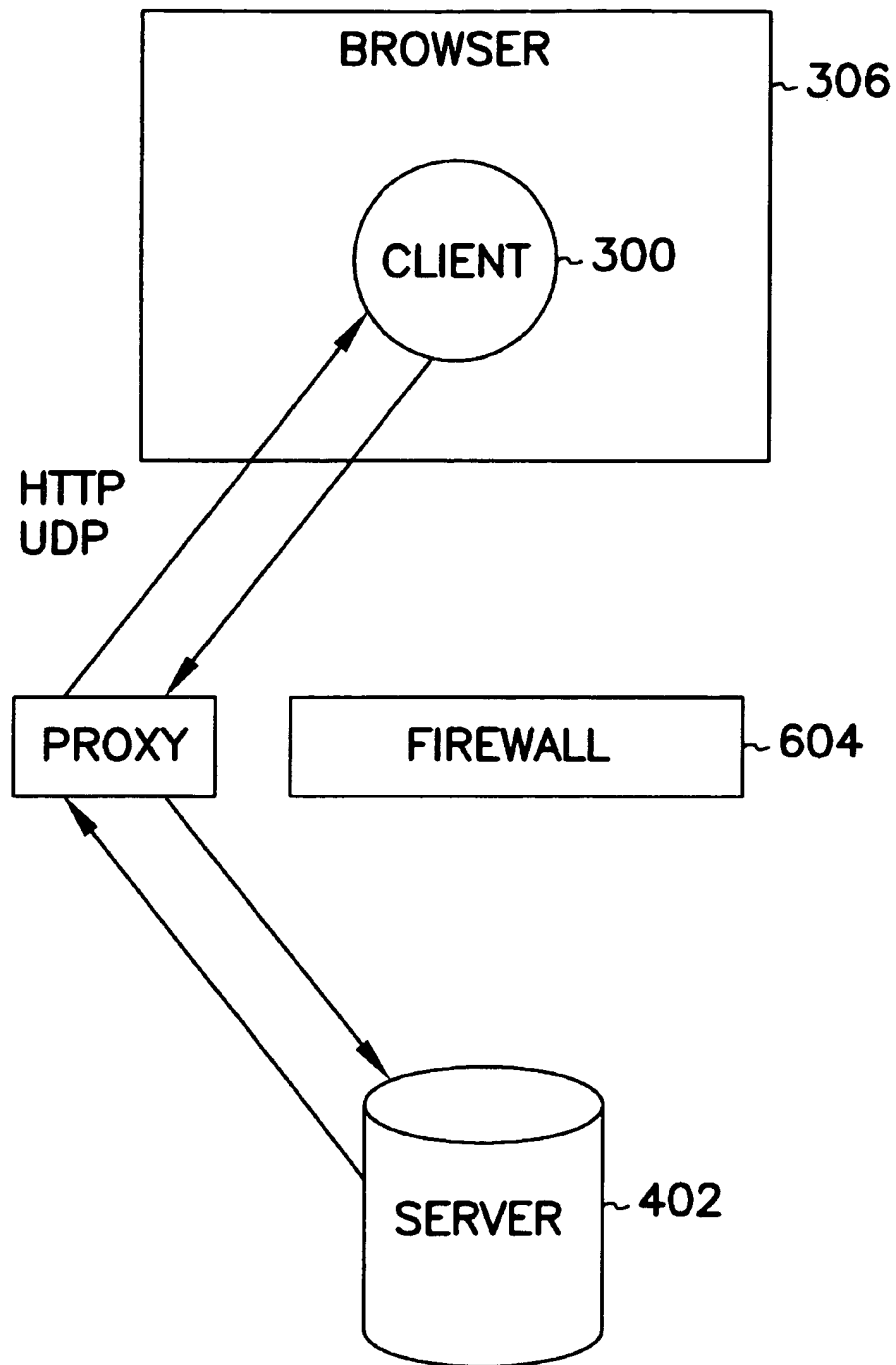


FIG. 5C

**FIG. 6**

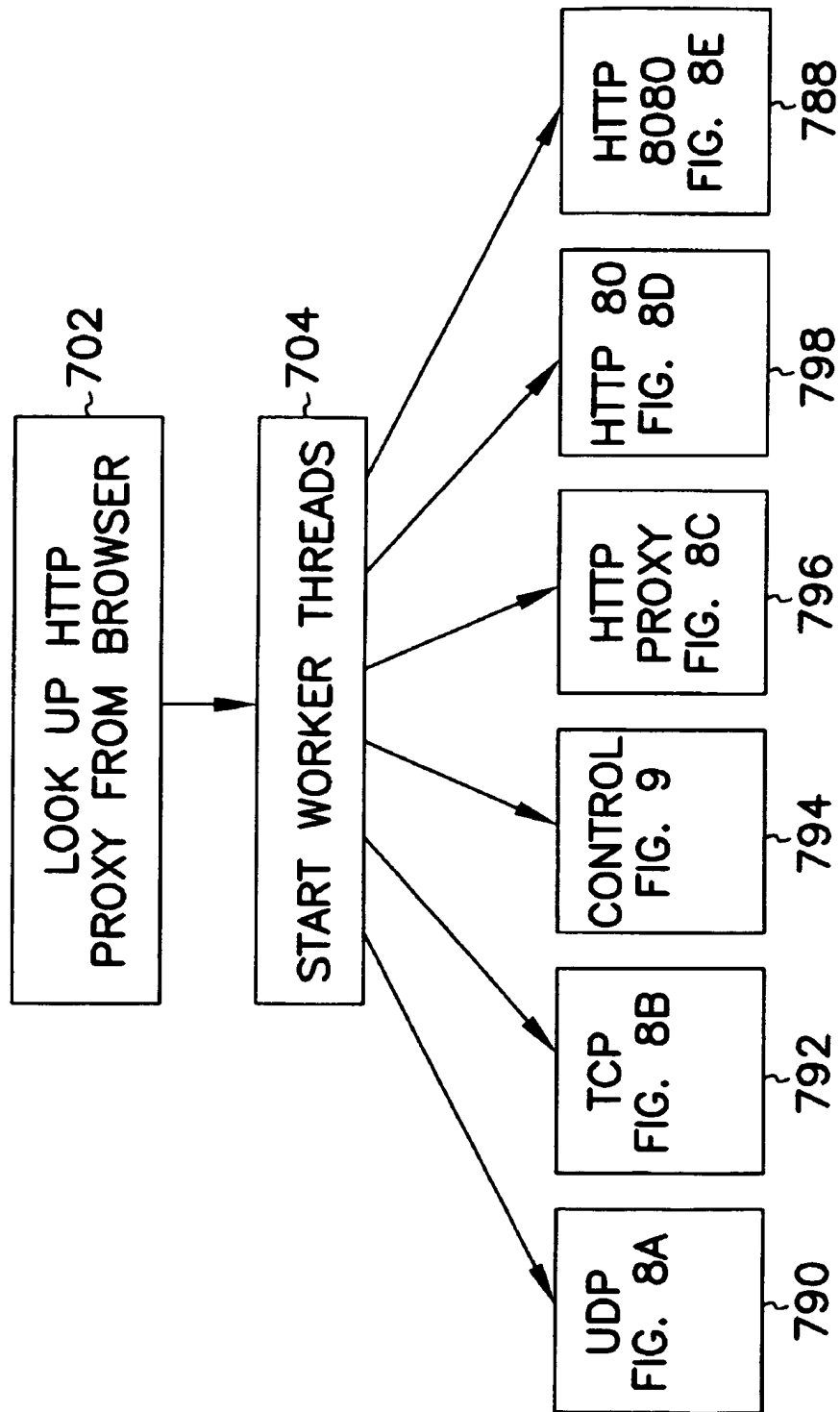


FIG. 7

FIG. 8A

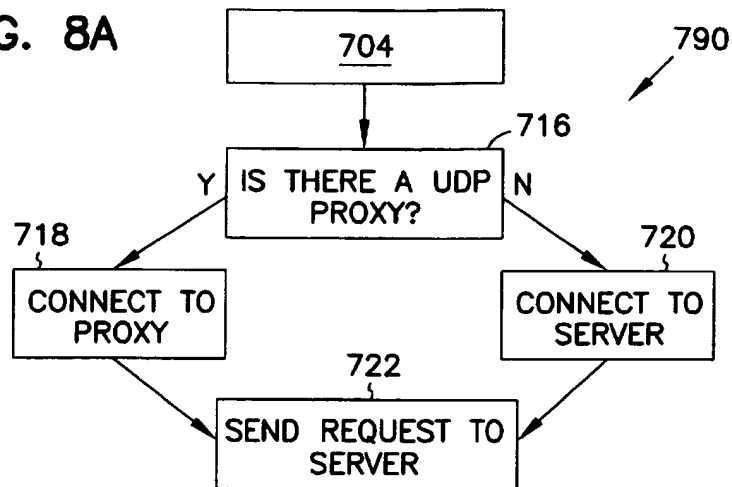


FIG. 8B

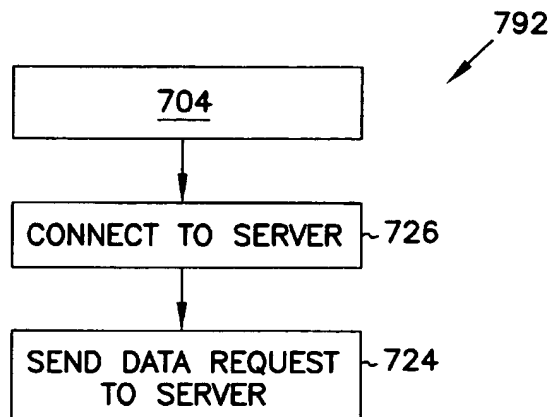


FIG. 8C

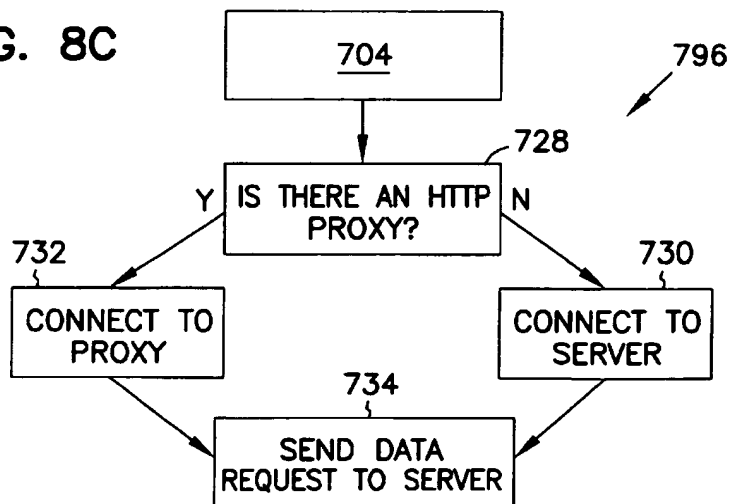


FIG. 8D

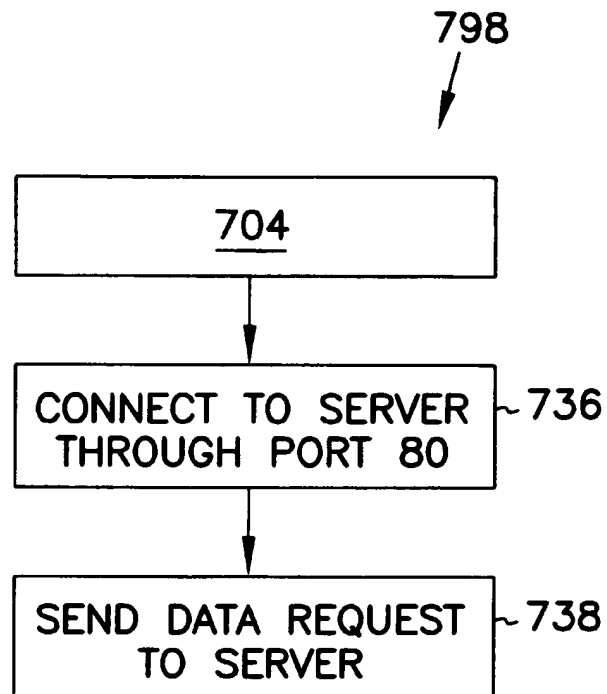
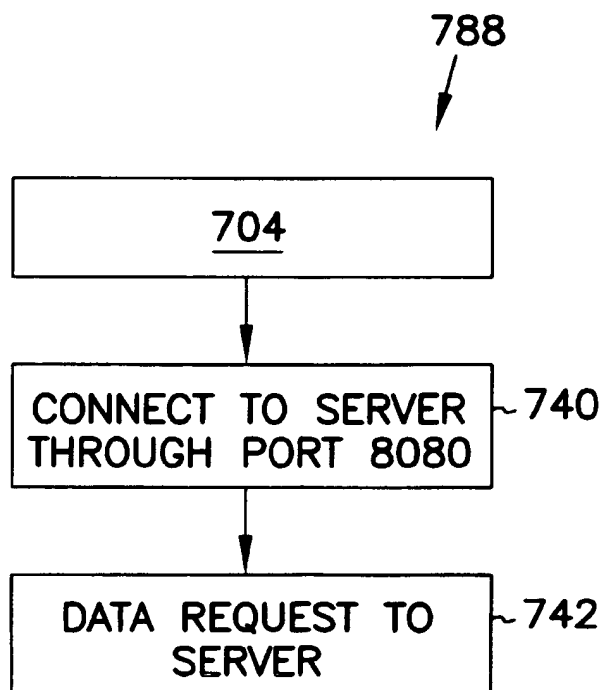


FIG. 8E



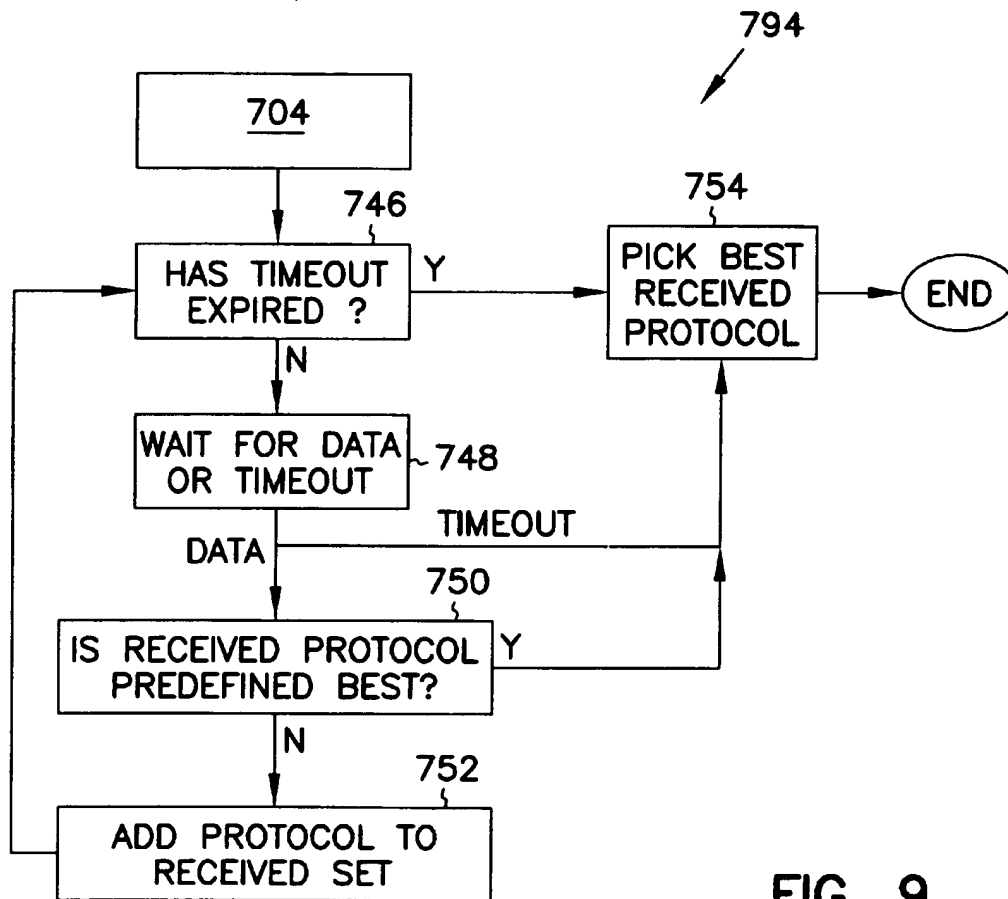


FIG. 9

**METHOD AND APPARATUS FOR
COMMUNICATION MEDIA COMMANDS
AND MEDIA DATA USING THE HTTP
PROTOCOL**

This application is related to co-pending U.S. application Ser. No. 08/818,805, filed on Mar. 14, 1997, entitled "Method and Apparatus for Implementing Motion Detection in Video Compression", U.S. application Ser. No. 08/819,507, filed on Mar. 14, 1997, entitled "Digital Video Signal Encoder and Encoding Method", U.S. application Ser. No. 08/818,804, filed on Mar. 14, 1997, entitled "Production of a Video Stream with Synchronized Annotations over a Computer Network", U.S. application Ser. No. 08/819,586, filed on Mar. 14, 1997, entitled "Method and Apparatus for Implementing Control Functions in a Streamed Video Display System", U.S. application Ser. No. 08/818,769, filed on Mar. 14, 1997, entitled "Method and Apparatus for Automatically Detecting Protocols in a Computer Network", U.S. application Ser. No. 08/818,127, filed on Mar. 14, 1997, entitled "Dynamic Bandwidth Selection for Efficient Transmission of Multimedia Streams in a Computer Network", U.S. application Ser. No. 08/819,585, filed on Mar. 14, 1997, entitled "Streaming and Display of a Video Stream with Synchronized Annotations over a Computer Network", U.S. application Ser. No. 08/818,664, filed on Mar. 14, 1997, entitled "Selective Retransmission for Efficient and Reliable Streaming of Multimedia Packets in a Computer Network", U.S. application Ser. No. 08/819,579, filed on Mar. 14, 1997, entitled "Method and Apparatus for Table-Based Compression with Embedded Coding", U.S. application Ser. No. 08/819,587, filed on Mar. 14, 1997, entitled "Method and Apparatus for Implementing Motion Estimation in Video Compression", U.S. application Ser. No. 08/818,826, filed on Mar. 14, 1997, entitled "Digital Video Signal Encoder and Encoding Method", all filed concurrently herewith, U.S. application Ser. No. 08/822,156, filed on Mar. 17, 1997, entitled "Method and Apparatus for Communication Media Commands and Data Using the HTTP Protocol", provisional U.S. application Ser. No. 60/036,662, filed on Jan. 30, 1997, entitled "Methods and Apparatus for Autodetecting Protocols in a Computer Network" U.S. application Ser. No. 08/625,650, filed on Mar. 29, 1996, entitled "Table-Based Low-Level Image Classification System", U.S. application Ser. No. 08/714,447, filed on Sep. 16, 1996, entitled "Multimedia Compression System with Additive Temporal Layers", and is a continuation-in-part of U.S. application Ser. No. 08/623,299, filed on Mar. 28, 1996, entitled "Table-Based Compression with Embedded Coding", which are all incorporated by reference in their entirety for all purposes.

BACKGROUND OF THE INVENTION

The present invention relates to data communication in a computer network. More particularly, the present invention relates to improved methods and apparatus for permitting a client computer in a client-server architecture computer network to exchange media commands and media data with the server using the HTTP (hypertext transfer protocol) protocol.

Client-server architectures are well known to those skilled in the computer art. For example, in a typical computer network, one or more client computers may be coupled to any number of server computers. Client computers typically refer to terminals or personal computers through which end users interact with the network. Server computers typically represent nodes in the computer network where data, application programs, and the like, reside. Server computers may

also represent nodes in the network for forwarding data, programs, and the likes from other servers to the requesting client computers.

To facilitate discussion, FIG. 1 illustrates a computer network 100, representing for example a subset of an international computer network popularly known as the Internet. As is well known, the Internet represents a well-known international computer network that links, among others, various military, governmental, educational, nonprofit, industrial and financial institutions, commercial enterprises, and individuals. There are shown in FIG. 1 a server 102, a server 104, and a client computer 106. Server computer 104 is separated from client computer 106 by a firewall 108, which may be implemented in either software or hardware, and may reside on a computer and/or circuit between client computer 106 and server computer 104.

Firewall 108 may be specified, as is well known to those skilled in the art, to prevent certain types of data and/or protocols from traversing through it. The specific data and/or protocols prohibited or permitted to traverse firewall 108 depend on the firewall parameters, which are typically set by a system administrator responsible for the maintenance and security of client computer 106 and/or other computers connected to it, e.g., other computers in a local area network. By way of example, firewall 108 may be set up to prevent TCP, UDP, or HTTP (Transmission Control Protocol, User Datagram Protocol, and Hypertext Transfer Protocol, respectively) data and/or other protocols from being transmitted between client computer 106 and server 104. The firewalls could be configured to allow specific TCP or UDP sessions, for example outgoing TCP connection to certain ports, UDP sessions to certain ports, and the like.

Without a firewall, any type of data and/or protocol may be communicated between a client computer and a server computer if appropriate software and/or hardware are employed. For example, server 102 resides on the same side of firewall 108 as client computer 106, i.e., firewall 108 is not disposed in between the communication path between server 102 and client computer 106. Accordingly, few, if any, of the protocols that client computer 106 may employ to communicate with server 102 may be blocked.

As is well known to those skilled in the art, some computer networks may be provided with proxies, i.e., software codes or hardware circuitries that facilitate the indirect communication between a client computer and a server around a firewall. With reference to FIG. 1, for example, client computer 106 may communicate with server 104 through proxy 120. Through proxy 120, HTTP data, which may otherwise be blocked by firewall 108 for the purpose of this example, may be transmitted between client computer 106 and server computer 104.

In the prior art, the HTTP protocol is typically employed to transmit web pages between the client computer and the server computer. As is well known to those skilled, the HTTP protocol, as specified by for example the Internet Request For Comments RFC 1945 (T. Berners-Lee et al.), typically defines only three types of requests to be sent from the client computer to the server, namely GET, POST, and HEAD. The POST command, for instance, is specified in RFC 1945 to be composed of a Request-Line, one or more Headers and Entity-Body.

In some cases, it may be desirable, however, to employ the HTTP protocol to communicate other types of commands and receive other types of data between the client computer and the server computer. By way of example, in applications such as real-time or live video streaming, the

HTTP protocol may represent, on some networks, the most advantageous protocol available for use in transmitting and receiving data. This is because, for example, there may exist firewalls or other network limitations that inhibit the use of other protocols for transmitting media control commands and for receiving media data. Media control commands may represent, for example, commands to fast forward on the play stream, to seek backward on the play stream, to begin playing at a certain frame, to stop, to pause, and the like. Media data may represent, for example, real-time or live video, audio, or annotation data. In these cases, the ability to use the HTTP protocol to transmit media commands and to receive media data may indeed be valuable.

In view of the foregoing, there are desired improved techniques for permitting a client computer in a client-server architecture computer network to exchange media commands and media data with the server computer using the HTTP (hypertext transfer protocol) protocol.

SUMMARY OF THE INVENTION

The invention relates, in one embodiment, to a method for employing a Hypertext Transfer Protocol (HTTP protocol) for transmitting streamed digital media data from a server. The server is configured for coupling to a client computer via a computer network. The method includes receiving at the server from the client an HTTP POST request. The POST request requests a first portion of the digital media data and includes a request header and a request entity-body. The request entity body includes a media command for causing the first portion of the digital media data to be sent from the server to the client. The method further includes sending an HTTP response to the client from the server. The HTTP response includes a response header and a response entity body. The response entity body includes at least a portion of the first portion of the digital media data.

In another embodiment, the invention relates to a computer readable medium containing computer readable instructions for transmitting streamed media data employing a Hypertext Transfer Protocol (HTTP protocol) from a server. The server is configured for coupling to a client computer via a computer network. The method includes receiving at the server from the client an HTTP POST request. The POST request requests a first portion of the digital media data and includes a request header and a request entity-body. The request entity body includes a media command for causing the first portion of the digital media data to be sent from the server to the client. The method further includes sending an HTTP response to the client from the server. The HTTP response includes a response header and a response entity body. The response entity body includes at least a portion of the first portion of the digital media data.

These and other features of the present invention will be described in more detail below in the detailed description of the invention and in conjunction with the following figures.

BRIEF DESCRIPTION OF THE DRAWINGS

To facilitate discussion, FIG. 1 illustrates a computer network, representing for example a portion of an international computer network popularly known as the Internet.

FIG. 2 is a block diagram of an exemplar computer system for carrying out the autodetect technique according to one embodiment of the invention.

FIG. 3 illustrates, in accordance with one embodiment, the control and data connections between a client application and a server computer when no firewall is provided in the network.

FIG. 4 illustrates another network arrangement wherein control and data connections are established through a firewall.

FIGS. 5A-B illustrates another network arrangement wherein media control commands and media data may be communicated between a client computer and a server computer using the HTTP protocol.

FIGS. 5C-D illustrate another network arrangement wherein multiple HTTP control and data connections are multiplexed through a single HTTP port.

FIG. 6 illustrates another network arrangement wherein control and data connections are transmitted between the client application and the server computer via a proxy.

FIG. 7 depicts, in accordance with one embodiment of the present invention, a simplified flowchart illustrating the steps of the inventive autodetect technique.

FIG. 8A depicts, in accordance with one aspect of the present invention, the steps involved in executing the UDP protocol thread of FIG. 7.

FIG. 8B depicts, in accordance with one aspect of the present invention, the steps involved in executing the TCP protocol thread of FIG. 7.

FIG. 8C depicts, in accordance with one aspect of the present invention, the steps involved in executing the HTTP protocol thread of FIG. 7.

FIG. 8D depicts, in accordance with one aspect of the present invention, the steps involved in executing the HTTP 80 protocol thread of FIG. 7.

FIG. 8E depicts, in accordance with one aspect of the present invention, the steps involved in executing the HTTP 8080 protocol thread of FIG. 7.

FIG. 9 illustrates, in accordance with one embodiment of the present invention, the steps involved in executing the control thread of FIG. 7.

DETAILED DESCRIPTION OF THE PREFERRED EMBODIMENTS

The present invention will now be described in detail with reference to a few preferred embodiments thereof as illustrated in the accompanying drawings. In the following description, numerous specific details are set forth in order to provide a thorough understanding of the present invention. It will be apparent, however, to one skilled in the art, that the present invention may be practiced without some or all of these specific details. In other instances, well known process steps have not been described in detail in order to not unnecessarily obscure the present invention.

In accordance with one aspect of the present invention, the client computer in a heterogeneous client-server computer network (e.g., client computer 106 in FIG. 1) is provided with an autodetect mechanism. When executed, the autodetect mechanism advantageously permits client computer 106 to select, in an efficient and automatic manner, the most advantageous protocol for communication between the client computer and its server. Once the most advantageous protocol is selected, parameters pertaining to the selected protocol are saved to enable the client computer, in future sessions, to employ the same selected protocol for communication.

In accordance with one particular advantageous embodiment, the inventive autodetect mechanism simultaneously employs multiple threads, through multiple connections, to initiate communication with the server computer, e.g., server 104. Each thread preferably employs a different protocol and requests the server computer to

5

respond to the client computer using the protocol associated with that thread. For example, client computer 106 may, employing the autodetect mechanism, initiate five different threads, using respectively the TCP, UDP, one of HTTP and HTTP proxy, HTTP through port (multiplex) 80, and HTTP through port (multiplex) 8080 protocols to request server 104 to respond.

Upon receiving a request, server 104 responds with data using the same protocol as that associated with the thread on which the request arrives. If one or more protocols is blocked and fails to reach server 104 (e.g., by a firewall), no response employing the blocked protocol would of course be transmitted from server 104 to client computer 106. Further, some of the protocols transmitted from server 104 to client computer 106 may be blocked as well. Accordingly, client computer may receive only a subset of the responses sent from server 104.

In one embodiment, client computer 106 monitors the set of received responses. If the predefined "best" protocol is received, that protocol is then selected for communication by client computer 106. The predefined "best" protocol may be defined in advance by the user and/or the application program. If the predefined "best" protocol is, however, blocked (as the request is transmitted from the client computer or as the response is transmitted from the server, for example) the most advantageous protocol may simply be selected from the set of protocols received back by the client computer. In one embodiment, the selection may be made among the set of protocols received back by the client computer within a predefined time period after the requests are sent out in parallel.

The selection of the most advantageous protocol for communication among the protocols received by client computer 106 may be performed in accordance with some predefined priority. For example, in the real-time data rendering application, the UDP protocol may be preferred over TCP protocol, which may be in turned preferred over the HTTP protocol. This is because UDP protocol typically can handle a greater data transmission rate and may allow client computer 106 to exercise a greater degree of control over the transmission of data packets.

HTTP data, while popular nowadays for use in transmitting web pages, typically involves a higher number of overhead bits, making it less efficient relative to the UDP protocol for transmitting real-time data. As is known, the HTTP protocol is typically built on top of TCP. The underlying TCP protocol typically handles the transmission and retransmission requests of individual data packets automatically. Accordingly, the HTTP protocol tends to reduce the degree of control client computer 106 has over the transmission of the data packets between server 104 and client computer 106. Of course other priority schemes may exist for different applications, or even for different real-time data rendering applications.

In one embodiment, as client computer 106 is installed and initiated for communication with server 104 for the first time, the autodetect mechanism is invoked to allow client computer 106 to send transmission requests in parallel (e.g., using different protocols over different connections) in the manner discussed earlier. After server 104 responds with data via multiple connections/protocols and the most advantageous protocol has been selected by client computer 106 for communication (in accordance with some predefined priority), the parameters associated with the selected protocol are then saved for future communication.

Once the most advantageous protocol is selected, the autodetect mechanism may be disabled, and future commu-

6

nication between client computer 106 and server 104 may proceed using the selected most advantageous protocol without further invocation of the autodetect mechanism. If the topology of computer network 100 changes and communication using the previously selected "most advantageous" protocol is no longer appropriate, the autodetect mechanism may be executed again to allow client computer 106 to ascertain a new "most advantageous" protocol for communication with server 104. In one embodiment, the user of client computer 106 may, if desired, initiate the autodetect mechanism at anytime in order to enable client computer 106 to update the "most advantageous" protocol for communication with server 104 (e.g., when the user of client computer 106 has reasons to suspect that the previously selected "most advantageous" protocol is no longer the most optimal protocol for communication).

The inventive autodetect mechanism may be implemented either in software or hardware, e.g., via an IC chip. If implemented in software, it may be carried out by any number of computers capable of functioning as a client computer in a computer network. FIG. 2 is a block diagram of an exemplar computer system 200 for carrying out the autodetect technique according to one embodiment of the invention. Computer system 200, or an analogous one, may be employed to implement either a client or a server of a computer network. The computer system 200 includes a digital computer 202, a display screen (or monitor) 204, a printer 206, a floppy disk drive 208, a hard disk drive 210, a network interface 212, and a keyboard 214. The digital computer 202 includes a microprocessor 216, a memory bus 218, random access memory (RAM) 220, read only memory (ROM) 222, a peripheral bus 224, and a keyboard controller 226. The digital computer 200 can be a personal computer (such as an Apple computer, e.g., an Apple Macintosh, an IBM personal computer, or one of the compatibles thereof), a workstation computer (such as a Sun Microsystems or Hewlett-Packard workstation), or some other type of computer.

The microprocessor 216 is a general purpose digital processor which controls the operation of the computer system 200. The microprocessor 216 can be a single-chip processor or can be implemented with multiple components. Using instructions retrieved from memory, the microprocessor 216 controls the reception and manipulation of input data and the output and display of data on output devices.

The memory bus 218 is used by the microprocessor 216 to access the RAM 220 and the ROM 222. The RAM 220 is used by the microprocessor 216 as a general storage area and as scratch-pad memory, and can also be used to store input data and processed data. The ROM 222 can be used to store instructions or program code followed by the microprocessor 216 as well as other data.

The peripheral bus 224 is used to access the input, output, and storage devices used by the digital computer 202. In the described embodiment, these devices include the display screen 204, the printer device 206, the floppy disk drive 208, the hard disk drive 210, and the network interface 212, which is employed to connect computer 200 to the network. The keyboard controller 226 is used to receive input from keyboard 214 and send decoded symbols for each pressed key to microprocessor 216 over bus 228.

The display screen 204 is an output device that displays images of data provided by the microprocessor 216 via the peripheral bus 224 or provided by other components in the computer system 200. The printer device 206 when operating as a printer provides an image on a sheet of paper or a

similar surface. Other output devices such as a plotter, typesetter, etc. can be used in place of, or in addition to, the printer device 206.

The floppy disk drive 208 and the hard disk drive 210 can be used to store various types of data. The floppy disk drive 208 facilitates transporting such data to other computer systems, and hard disk drive 210 permits fast access to large amounts of stored data.

The microprocessor 216 together with an operating system operate to execute computer code and produce and use data. The computer code and data may reside on the RAM 220, the ROM 222, the hard disk drive 220, or even on another computer on the network. The computer code and data could also reside on a removable program medium and loaded or installed onto the computer system 200 when needed. Removable program mediums include, for example, CD-ROM, PC-CARD, floppy disk and magnetic tape.

The network interface circuit 212 is used to send and receive data over a network connected to other computer systems. An interface card or similar device and appropriate software implemented by the microprocessor 216 can be used to connect the computer system 200 to an existing network and transfer data according to standard protocols.

The keyboard 214 is used by a user to input commands and other instructions to the computer system 200. Other types of user input devices can also be used in conjunction with the present invention. For example, pointing devices such as a computer mouse, a track ball, a stylus, or a tablet can be used to manipulate a pointer on a screen of a general-purpose computer.

The invention can also be embodied as computer readable code on a computer readable medium. The computer readable medium is any data storage device that can store data which can be thereafter be read by a computer system. Examples of the computer readable medium include read-only memory, random-access memory, CD-ROMs, magnetic tape, optical data storage devices. The computer readable code can also be distributed over a network coupled computer systems so that the computer readable code is stored and executed in a distributed fashion.

FIGS. 3-6 below illustrate, to facilitate discussion, some possible arrangements for the transmission and receipt of data in a computer network. The arrangements differ depend on which protocol is employed and the configuration of the network itself. FIG. 3 illustrates, in accordance with one embodiment, the control and data connections between a client application 300 and server 302 when no firewall is provided in the network.

Client application 300 may represent, for example, the executable codes for executing a real-time data rendering program such as the Web Theater Client 2.0, available from Vxtreme, Inc. of Sunnyvale, Calif. In the example of FIG. 3, client application 300 includes the inventive autodetect mechanism and may represent a plug-in software module that may be installed onto a browser 306. Browser 306 may represent, for example, the application program which the user of the client computer employs to navigate the network. By way of example, browser 306 may represent one of the popular Internet browser programs, such as Netscape™ by Netscape Communications Inc. of Mountain View, Calif. or Microsoft Explorer by Microsoft Corporation of Redmond, Wash.

When the autodetect mechanism of client application 300 is executed in browser 306 (e.g., during the set up of client application 300), client application 300 sends a control request over control connection 308 to server 302. Although

multiple control requests are typically sent in parallel over multiple control connections using different protocols as discussed earlier, only one control request is depicted in FIG. 3 to facilitate ease of illustration.

The protocol employed to send the control request over control connection 308 may represent, for example, TCP, or HTTP. If UDP protocol is requested from the server, the request from the client may be sent via the control connection using for example the TCP protocol. Initially, each control request from client application 300 may include, for example, the server name that identifies server 302, the port through which control connection may be established, and the name of the video stream requested by client application 300. Server 302 then responds with data via data connection 310.

In FIG. 3, it is assumed that no proxies and/or firewalls exist. Accordingly, server 302 responds using the same protocol as that employed in the request. If the request employs TCP, however, server 302 may attempt to respond using either UDP or TCP data connections (depending on the specifics of the request). The response is sent to client application via data connection 310. If the protocol received by the client application is subsequently selected to be the "most advantageous" protocol, subsequent communication between client application 300 and server 302 may take place via control connection 308 and data connection 310. Subsequent control requests sent by client application 300 via control connection 308 may include, for example, stop, play, fast forward, rewind, pause, unpause, and the like. These control requests may be utilized by server 302 to control the delivery of the data stream from server 302 to client application 300 via data connection 310.

It should be noted that although only one control connection and one data connection is shown in FIG. 3 to simplify illustration, multiple control and data connections utilizing the same protocol may exist during a data rendering session. Multiple control and data connections may be required to handle the multiple data streams (e.g., audio, video, annotation) that may be needed in a particular data rendering session. If desired, multiple clients applications 300 may be installed within browser 306, e.g., to simultaneously render multiple video clips, each with its own sound and annotations.

FIG. 4 illustrates another network arrangement wherein control and data connections are established through a firewall. As mentioned earlier, a firewall may have policies that restrict or prohibit the traversal of certain types of data and/or protocols. In FIG. 4, a firewall 400 is disposed between client application 300 and server 402. Upon execution, client application 300 sends control request using a given protocol via firewall 400 to server 402. Server 402 then responds with data via data connection 410, again via firewall 400.

If the data and/or protocol can be received by the client computer through firewall 400, client application 300 may then receive data from server 402 (through data connection 408) in the same protocol used in the request. As before, if the request employs the TCP protocol, the server may respond with data connections for either TCP or UDP protocol (depending on the specifics of the request). Protocols that may traverse a firewall may include one or more of the following: UDP, TCP, and HTTP.

In accordance with one aspect of the present invention, the HTTP protocol may be employed to send/receive media data (video, audio, annotation, or the like) between the client and the server. FIG. 5A is a prior art drawing illustrating how

a client browser may communicate with a web server using a port designated for communication. In FIG. 5, there is shown a web server 550, representing the software module for serving web pages to a browser application 552. Web server 550 may be any of the commercially available web servers that are available from, for example, Netscape Communications Inc. of Mountain View, Calif. or Microsoft Corporation of Redmond, Wash. Browser application 552 represents for example the Netscape browser from the aforementioned Netscape Communications, Inc., or similarly suitable browser applications.

Through browser application 552, the user may, for example, obtain web pages pertaining to a particular entity by sending an HTTP request (e.g., GET) containing the URL (uniform resource locator) that identifies the web page file. The request sent via control connection 553 may arrive at web server 550 through the HTTP port 554. HTTP port 554 may represent any port through which HTTP communication is enabled. HTTP port 554 may also represent the default port for communicating web pages with client browsers. The HTTP default port may represent, for example, either port 80 or port 8080 on web server 550. As is known, one or both of these ports on web server 550 may be available for web page communication even if there are firewalls disposed between the web server 550 and client browser application 552, which otherwise block all HTTP traffic in other ports. Using the furnished URL, web server 550 may then obtain the desired web page(s) for sending to client browser application 552 via data connection 556.

The invention, in one embodiment, involves employing the HTTP protocol to communicate media commands from a browser application or browser plug-in to the server. Media commands are, for example, PLAY, STOP, REWIND, FAST FORWARD, and PAUSE. The server computer may represent, for example, a web server. The server computer may also represent a video server for streaming video to the client computer. Through the use of the HTTP protocol the client computer may successfully send media control requests and receive media data through any HTTP port. If the default HTTP port, e.g., port 80 or 8080, is specified, the client may successfully send media control requests and receive media data even if there exists a firewall or an HTTP Proxy disposed in between the server computer and the client computer, which otherwise blocks all other traffic that does not use the HTTP protocol. For example, these firewalls or HTTP Proxies do not allow regular TCP or UDP packets to go through.

As is well known to those skilled, the HTTP protocol, as specified by for example the Internet Request For Comments RFC 1945 (T. Berners-Lee et al.), typically defines only three types of requests to be sent from the client computer to the server, namely GET, POST, and HEAD. The POST command, for instance, is specified in RFC 1945 to be composed of a Request-Line, one or more Headers and Entity-Body. To send media commands like PLAY, REWIND, etc., the invention in one embodiment sends the media command as part of the Entity-Body of the HTTP POST command. The media command can be in any format or protocol, and can be, for instance, in the same format as that employed when firewalls are not a concern and plain TCP protocol can be used. This format can be, for example, RTSP (Real Time Streaming Protocol).

When a server gets an HTTP request, it answers the client with an HTTP Response. Responses are typically composed of a Status-Line, one or more headers, and an Entity-Body. In one embodiment of this invention, the response to the media commands is sent as the Entity-Body of the response to the original HTTP request that carried the media command.

FIG. 5B illustrates this use of HTTP for sending arbitrary media commands. In FIG. 5B, the plug-in application 560 within client browser application 562 may attempt to receive media data (e.g., video, audio, annotation, or the like) by first sending an HTTP request to server 564 via control connection 565. For example, a REWIND command could be sent from the client 560 to the server 564 as an HTTP packet 570 of the form: "POST/HTTP/1.0<Entity-Body containing rewind command in any suitable media protocol>". The server can answer to this request with an HTTP response 572 of the form: "HTTP/1.0 200ok<Entity-Body containing rewind response in any suitable media protocol>".

The HTTP protocol can be also used to send media data across firewalls. The client can send a GET request to the video server, and the video server can then send the video data as the Entity-Body of the HTTP response to this GET request.

Some firewalls may be restrictive with respect to HTTP data and may permit HTTP packets to traverse only on a certain port, e.g., port 80 and/or port 8080. FIG. 5C illustrates one such situation. In this case, the control and data communications for the various data stream, e.g., audio, video, and/or annotation associated with different rendering sessions (and different clients) may be multiplexed using conventional multiplexer code and/or circuit 506 at client application 300 prior to being sent via port 502 (which may represent, for example, HTTP port 80 or HTTP port 8080). The inventive combined use of the HTTP protocol and of the multiplexer for transmitting media control and data is referred to as the HTTP multiplex protocol, and can be used to send this data across firewalls that only allow HTTP traffic on specific ports, e.g., port 80 or 8080.

At server 402, representing, for example, server 104 of FIG. 1, conventional demultiplexer code and/or circuit 508 may be employed to decode the received data packets to identify which stream the control request is associated with. Likewise, data sent from server 402 to client application 300 may be multiplexed in advance at server 402 using for example conventional multiplexer code and/or circuit 510. The multiplexed data is then sent via port 502. At client application 300, the multiplexed data may be decoded via conventional demultiplexer code and/or circuit 512 to identify which stream the received data packets is associated with (audio, video, or annotation).

Multiplexing and demultiplexing at the client and/or server may be facilitated for example by the use of the Request-URL part of the Request-Line of HTTP requests. As mentioned above, the structure of HTTP requests is described in RFC 1945. The Request-URL may, for example, identify the stream associated with the data and/or control request being transmitted. In one embodiment, the additional information in the Request-URL in the HTTP header may be as small as one or a few bits added to the LHTTP request sent from client application 300 to server 402.

To further facilitate discussion of the inventive HTTP multiplexing technique, reference may now be made to FIG. 5D. In FIG. 5D, the plug-in application 660 within client plug-in application 660 may attempt to receive media data (e.g., video, audio, annotation, or the like) by first sending a control request 670 to server 664 via control connection 665. The control request is an HTTP request, which arrives at the HTTP default port 654 on server 664. As mentioned earlier, the default HTTP port may be either port 80 or port 8080 in one embodiment.

In one example, the control request 670 from client plug-in 660 takes the form of a command to "POST/12469

11

HTTP/1.0<Entity-Body>" which indicates to the server (through the designation 12469 as the Request-URL) that this is a control connection. The Entity-Body contains, as described above, binary data that informs the video server that the client plug-in 660 wants to display a certain video or audio clip. Software codes within server 664 may be employed to assign a unique ID to this particular request from this particular client.

For discussion sake, assume that server 664 associates unique ID 35,122 with a video data connection between itself and client plug-in application 660, and unique ID 29,999 with an audio data connection between itself and client plug-in application. The unique ID is then communicated as message 672 from server 664 to client plug-in application 660, again through the aforementioned HTTP default port using data connection 667. The Entity-Body of message 672 contains, among other things and as depicted in detail 673, the audio and/or video session ID. Note that the unique ID is unique to each data connection (e.g., each of the audio, video, and annotation connections) of each client plug-in application (since there may be multiple client plug-in applications attempting to communicate through the same port).

Once the connection is established, the same unique ID number is employed by the client to issue HTTP control requests to server 664. By way of example, client plug-in application 660 may issue a command "GET/35,122 HTTP/1.0" or "POST/35,122 HTTP/1.0<Entity-Body containing binary data with the REWIND media command>" to request a video file or to rewind on the video file. Although the rewind command is used in FIGS. 5A-5D to facilitate ease of discussion, other media commands, e.g., fast forward, pause, real-time play, live-play, or the like, may of course be sent in the Entity-Body. Note that the unique ID is employed in place of or in addition to the Request-URL to qualify the Request-URL.

Once the command is received by server 664, the unique ID number (e.g. 35,122) may be employed by the server to demultiplex the command to associate the command with a particular client and data file. This unique ID number can also attach to the HTTP header of HTTP responses sent from server 664 to client plug-in application 660, through the same HTTP default port 654 on server 664, to permit client plug-in application 660 to ascertain whether an HTTP data packet is associated with a given data stream.

Advantageously, the invention permits media control commands and media data to be communicated between the client computer and the server computer via the default HTTP port, e.g., port 80 or 8080 in one embodiment, even if HTTP packets are otherwise blocked by a firewall disposed between the client computer and the server computer. The association of each control connection and data connection to each client with a unique ID advantageously permits multiple control and data connections (from one or more clients) to be established through the same default HTTP port on the server, advantageously bypassing the firewall. Since both the server and the client have the demultiplexer code and/or circuit that resolve a particular unique ID into a particular data stream, multiplexed data communication is advantageously facilitated thereby.

In some networks, it may not be possible to traverse the firewall due to stringent firewall policies. As mentioned earlier, it may be possible in these situations to allow the client application to communicate with a server using a proxy. FIG. 6 illustrates this situation wherein client application 300 employs proxy 602 to communicate with server

12

402. The use of proxy 602 may be necessary since client application 300 may employ a protocol which is strictly prohibited by firewall 604. The identity of proxy 602 may be found in browser program 306, e.g., Netscape as it employs the proxy to download its web pages, or may be configured by the user himself. Typical protocols that may employ a proxy for communication, e.g., proxy 602, includes HTTP and UDP.

In accordance with one embodiment of the present invention, the multiple protocols that may be employed for communication between a server computer and a client computer are tried in parallel during autodetect. In other words, the connections depicted in FIGS. 3, 4, 5C, and 6 may be attempted simultaneously and in parallel over different control connections by the client computer. Via these control connections, the server is requested to respond with various protocols.

If the predefined "best" protocol (predetermined in accordance with some predefined protocol priority) is received by the client application from the server, autodetect may, in one embodiment, end immediately and the "best" protocol is selected for immediate communication. In one real-time data rendering application, UDP is considered the "best" protocol, and the receipt of UDP data by the client may trigger the termination of the autodetect.

If the "best" protocol has not been received after a predefined time period, the most advantageous protocol (in terms of for example data transfer rate and/or transmission control) is selected among the set of protocols received by the client. The selected protocol may then be employed for communication between the client and the server.

FIG. 7 depicts, in accordance with one embodiment of the present invention, a simplified flowchart illustrating the steps of the inventive autodetect technique. In FIG. 7, the client application starts (in step 702) by looking up the HTTP proxy, if there is any, from the browser. As stated earlier, the client computer may have received a web page from the browser, which implies that the HTTP protocol may have been employed by the browser program for communication. If a HTTP proxy is required, the name and location of the HTTP proxy is likely known to the browser, and this knowledge may be subsequently employed by the client to at least enable communication with the server using the HTTP proxy protocol, i.e., if a more advantageous protocol cannot be ascertained after autodetect.

In step 704, the client begins the autodetect sequence by starting in parallel the control thread 794, along with five protocol threads 790, 792, 796, 798, and 788. As the term is used herein, parallel refers to both the situation wherein the multiple protocol threads are sent parallelly starting at substantially the same time (having substantially similar starting time), and the situation wherein the multiple protocol threads simultaneously execute (executing at the same time), irrespective when each protocol thread is initiated. In the latter case, the multiple threads may have, for example, staggered start time and the initiation of one thread may not depend on the termination of another thread.

Control thread 794 represents the thread for selecting the most advantageous protocol for communication. The other protocol threads 790, 792, 796, 798, and 788 represent threads for initiating in parallel communication using the various protocols, e.g., UDP, TCP, HTTP proxy, HTTP through port 80 (HTTP 80), and HTTP through port 8080 (HTTP 8080). Although only five protocol threads are shown, any number of protocol threads may be initiated by the client, using any conventional and/or suitable protocols.

13

The steps associated with each of threads 794, 790, 792, 796, 798, and 788 are discussed herein in connection with FIGS. 8A-8E and 9.

In FIG. 8A, the UDP protocol thread is executed. The client inquires in step 716 whether there requires a UDP proxy. If the UDP proxy is required, the user may obtain the name of the UDP proxy from, for example, the system administrator in order to use the UDP proxy to facilitate communication to the proxy (in step 718). If no UDP proxy is required, the client may directly connect to the server (in step 720). Thereafter, the client may begin sending a data request (i.e., a control request) to the server in step 722 using the UDP protocol (either through the proxy if a proxy is involved or directly to the server if no proxy is required).

In FIG. 8B, the TCP protocol thread is executed. If TCP protocol is employed, the client typically directly connects to the server (in step 726). Thereafter, the client may begin sending a data request (i.e., a control request) to the server using the TCP protocol (step 724).

In FIG. 8C, the HTTP protocol thread is executed. The client inquires in step 716 whether there requires a HTTP proxy. If the HTTP proxy is required, the user may obtain the name of the HTTP proxy from, for example, the browser since, as discussed earlier, the data pertaining to the proxy may be kept by the browser. Alternatively, the user may obtain data pertaining to the HTTP proxy from the system administrator in order to use the HTTP proxy to facilitate communication to the server (in step 732).

If no HTTP proxy is required, the client may directly connect to the server (in step 730). Thereafter, the client may begin sending a data request (i.e., a control request) to the server in step 734 using the HTTP protocol (either through the proxy if a proxy is involved or directly to the server if no proxy is required).

In FIG. 8D, the HTTP 80 protocol thread is executed. If HTTP 80 protocol is employed, HTTP data may be exchanged but only through port 80, which may be for example the port on the client computer through which communication with the network is permitted. Through port 80, the client typically directly connects to the server (in step 736). Thereafter, the client may begin sending a data request (i.e., a control request) to the server (step 738) using the HTTP 80 protocol.

In FIG. 8E, the HTTP 8080 protocol thread is executed. If HTTP 8080 protocol is employed, HTTP data may be exchanged but only through port 8080, which may be the port on the client computer for communicating with the network. Through port 8080, the client typically directly connects to the server (in step 740). Thereafter, the client may begin sending a data request (i.e., a control request) to the server (step 742) using the HTTP 8080 protocol. The multiplexing and demultiplexing techniques that may be employed for communication through port 8080, as well as port 80 of FIG. 8D, have been discussed earlier and are not repeated here for brevity sake.

FIG. 9 illustrates, in accordance with one embodiment of the present invention, control thread 794 of FIG. 7. It should be emphasized that FIG. 7 is but one way of implementing the control thread; other techniques of implementing the control thread to facilitate autodetect should be apparent to those skilled in the art in view of this disclosure. In step 746, the thread determines whether the predefined timeout period has expired. The predefined timeout period may be any predefined duration (such as 7 seconds for example) from the time the data request is sent out to the server (e.g., step 722 of FIG. 8A). In one embodiment, each protocol thread

14

has its own timeout period whose expiration occurs at the expiration of a predefined duration after the data request using that protocol has been sent out. When all the timeout periods associated with all the protocols have been accounted for, the timeout period for the autodetect technique is deemed expired.

If the timeout has occurred, the thread moves to step 754 wherein the most advantageous protocol among the set of protocols received back from the server is selected for communication. As mentioned, the selection of the most advantageous protocol may be performed in accordance with some predefined priority scheme, and data regarding the selected protocol may be saved for future communication sessions between this server and this client.

If no timeout has occurred, the thread proceeds to step 748 to wait for either data from the server or the expiration of the timeout period. If timeout occurs, the thread moves to step 754, which has been discussed earlier. If data is received from the server, the thread moves to step 750 to ascertain whether the protocol associated with the data received from the server is the predefined "best" protocol, e.g., in accordance with the predefined priority.

If the predefined "best" protocol (e.g., UDP in some real-time data rendering applications) is received, the thread preferably moves to step 754 to terminate the autodetect and to immediately begin using this protocol for data communication instead of waiting of the timeout expiration. Advantageously, the duration of the autodetect sequence may be substantially shorter than the predefined timeout period. In this manner, rapid autodetect of the most suitable protocol and rapid establishment of communication are advantageously facilitated.

If the predefined "best" protocol is not received in step 750, the thread proceeds to step 752 to add the received protocol to the received set. This received protocol set represents the set of protocols from which the "most advantageous" (relatively speaking) protocol is selected. The most advantageous protocol is ascertained relative to other protocols in the received protocol set irrespective whether it is the predefined "best" protocol in accordance with the predefined priority. As an example of a predefined protocol priority, UDP may be deemed to be best (i.e., the predefined best), followed by TCP, HTTP, then HTTP 80 and HTTP 8080 (the last two may be equal in priority). As mentioned earlier, the most advantageous protocol is selected from the received protocol set preferably upon the expiration of the predefined timeout period.

From step 752, the thread returns to step 746 to test whether the timeout period has expired. If not, the thread continues along the steps discussed earlier.

Note that since the invention attempts to establish communication between the client application and the server computer in parallel, the time lag between the time the autodetect mechanism begins to execute and the time when the most advantageous protocol is determined is minimal. If communication attempts have been tried in serial, for example, the user would suffer the delay associated with each protocol thread in series, thereby disadvantageously lengthening the time period between communication attempt and successful establishment of communication.

The saving in time is even more dramatic in the event the network is congested or damaged. In some networks, it may take anywhere from 30 to 90 seconds before the client application realizes that an attempt to connect to the server (e.g., step 720, 726, 730, 736, or 740) has failed. If each protocol is tried in series, as is done in one embodiment, the

15

delay may, in some cases, reach minutes before the user realizes that the network is unusable and attempts should be made at a later time.

By attempting to establish communication via the multiple protocols in parallel, network-related delays are suffered in parallel. Accordingly, the user does not have to wait for multiple attempts and failures before being able to ascertain that the network is unusable and an attempt to establish communication should be made at a later time. In one embodiment, once the user realizes that all parallel attempts to connect with the network and/or the proxies have failed, there is no need to make the user wait until the expiration of the timeout periods of each thread. In accordance with this embodiment, the user is advised to try again as soon as it is realized that parallel attempts to connect with the server have all failed. In this manner, less of the user's time is needed to establish optimal communication with a network.

While this invention has been described in terms of several preferred embodiments, there are alterations, permutations, and equivalents which fall within the scope of this invention. For example, although the invention has been described with reference with sending out protocol threads in parallel, the automatic protocol detection technique also applies when the protocol threads are sent serially. In this case, while it may take longer to select the most advantageous protocol for selection, the automatic protocol detection technique accomplishes the task without requiring any sophisticated technical knowledge on the part of the user of the client computer. The duration of the autodetect technique, even when serial autodetect is employed, may be shortened by trying the protocols in order of their desirability and ignoring less desirable protocols once a more desirable protocol is obtained. It should also be noted that there are many alternative ways of implementing the methods and apparatuses of the present invention. It is therefore intended that the following appended claims be interpreted as including all such alterations, permutations, and equivalents as fall within the true spirit and scope of the present invention.

What is claimed is:

1. A method for employing a Hypertext Transfer Protocol (HTTP protocol) for transmitting streamed digital media data from a server, said server being configured for coupling to a client via computer network, comprising:

receiving at said server from said client an HTTP POST request, said POST request requesting a first portion of said digital media data and including a request header and a request entity-body, said request entity body including a media command for causing said first portion of said digital media data to be sent from said server to said client; and

sending an HTTP response to said client from said server, said HTTP response including a response header and a response entity body, said response entity body including at least a portion of said first portion of said digital media data, wherein said digital media data includes video data.

2. A method for employing a Hypertext Transfer Protocol (HTTP protocol) for transmitting streamed digital media data from a server, said server being configured for coupling to a client via computer network, comprising:

receiving at said server from said client an HTTP POST request, said POST request requesting a first portion of said digital media data and including a request header and a request entity-body, said request entity body including a media command for causing said first

16

portion of said digital media data to be sent from said server to said client; and sending an HTTP response to said client from said server, said HTTP response including a response header and a response entity body, said response entity body including at least a portion of said first portion of said digital media data,

wherein said digital media data represents live video data, said live video data representing data to be rendered at said client as an event related to said live video data is being recorded.

3. The method of claim 2 wherein said media command includes a REWIND command and said first portion of said digital media data represents digital media data that is earlier in time than a video frame currently displayed at said client.

4. The method of claim 2 wherein said media command includes a FAST FORWARD command and said first portion of said digital media data represents digital media data that is later in time than a video frame currently displayed at said client.

5. The method of claim 2 wherein said media command includes a LIVE PLAY command and said first portion of said digital media data represents digital media data that is recorded live.

6. The method of claim 2 wherein said media command includes a REAL-TIME PLAY command and said first portion of said digital media data represents digital media data that is stored earlier and streamed through said server.

7. The method of claim 2 wherein said HTTP protocol is permitted on only a selected port of said server, said HTTP port request and said HTTP response being multiplexed through said selected port.

8. The method of claim 7 wherein said selected port represents one of a port 80 and a port 8080 on said server.

9. The method of claim 8 wherein said server is employed to receive a plurality of HTTP requests from other clients coupled to said server, said plurality of HTTP requests also being multiplexed via said selected port.

10. The method of claim 9 wherein said HTTP requests includes a unique ID, said unique ID identifying said digital media data as digital media requested by said client, said unique ID being assigned by said server when said client establishes connection with said server.

11. A computer readable medium containing readable instructions for transmitting streamed media data employing a Hypertext Transfer Protocol (HTTP protocol) from a server, said server being configured for coupling to a client via a computer network, said computer readable instructions implementing the steps of:

receiving at said server from said client an HTTP POST request, said POST request requesting a first portion of said digital media data and including a request header and a request entity-body, said request entity body including a media command for causing said first portion of said digital media data to be sent from said server to said client; and

sending an HTTP response to said client from said server, said HTTP response including a response header and a response entity body, said response entity body including at least a portion of said digital media data, wherein said digital media data includes video data.

12. A computer readable medium containing readable instructions for transmitting streamed media data employing a Hypertext Transfer Protocol (HTTP protocol) from a server, said server being configured for coupling to a client via a computer network, said computer readable instructions implementing the steps of:

receiving at said server from said client an HTTP POST request, said POST request requesting a first portion of

17

said digital media data and including a request header and a request entity-body, said request entity body including a media command for causing said first portion of said digital media data to be sent from said server to said client; and

5 sending an HTTP response to said client from said server, said HTTP response including a response header and a response entity body, said response entity body including at least a portion of said digital media data,

10 wherein said digital media data represents live video data, said live video data representing data to be rendered at said client as an event related to said live video data is being recorded.

13. The computer readable medium of claim 12 wherein said media command includes a REWIND command and said first portion of said digital media data represents digital media data that is earlier in time than a video frame currently displayed at said client.

14. The computer readable medium of claim 12 wherein said media command includes a FAST FORWARD command and said first portion of said digital media data represents digital media data that is later in time than a video frame currently displayed at said client.

15 15. The computer readable medium of claim 12 wherein said media command includes a LIVE PLAY command and said first portion of said digital media data represents digital media data that is recorded live.

16. The computer readable medium of claim 12 wherein said media command includes a REAL-TIME PLAY command and said first portion of said digital media data represents digital media data that is stored earlier and streamed through said server.

17. The computer readable medium of claim 12 wherein said HTTP protocol is permitted on only a selected port of said server, said HTTP port request and said HTTP response being multiplexed through said selected port.

18. The computer readable medium of claim 17 wherein said selected port represents one of a port 80 and a port 8080 on said server.

19. The computer readable medium of claim 18 wherein said server is employed to receive a plurality of HTTP requests from other clients coupled to said server, said plurality of HTTP requests also being multiplexed via said selected port.

20. The computer readable medium of claim 19 wherein said HTTP requests includes a unique ID, said unique ID identifying said digital media data as digital media requested by said client, said unique ID being assigned by said server when said client establishes connection with said server.

21. A method for employing a Hypertext Transfer Protocol (HTTP protocol) for transmitting streamed digital media data from a server, said server being configured for coupling to a client via computer network, comprising:

50 receiving at said server from said client an HTTP POST request, said POST request requesting a first portion of said digital media data and including a request header and a request entity-body, said request entity body including a media command for causing said first portion of said digital media data to be sent from said server to said client; and

55 sending an HTTP response to said client from said server, said HTTP response including a response header and a response entity body said response entity body including at least a portion of said first portion of said digital media data, wherein the digital media data comprises audio data.

22. A computer readable medium containing readable instructions for transmitting streamed media data employing

18

a Hypertext Transfer Protocol (HTTP protocol) form a server, said server being configured for coupling to a client via a computer network, said computer readable instructions implementing the steps of:

5 receiving at said server from said client an HTTP POST request, said POST request requesting a first portion of said digital media data and including a request header and a request entity-body, said request entity body including a media command for causing said first portion of said digital media data to be sent from said server to said client; and

sending an HTTP response to said client from said server, said HTTP response including a response header and a response entity body said response entity body including at least a portion of said digital media data, wherein the digital media data comprises audio data.

23. A system comprising:

a client sending a Hypertext Transfer Protocol (HTTP protocol) POST request requesting a first portion of a digital media data and including a media command for causing the first portion of the digital media data to be sent from the server to the client; and

a server coupled to the client, wherein the server is sending an HTTP response to the client, wherein the HTTP response has at least a portion of the first portion of the digital media data, wherein said digital media data includes video data.

24. A system comprising:

a client sending a Hypertext Transfer Protocol (HTTP protocol) POST request requesting a first portion of a digital media data and including a media command for causing the first portion of the digital media data to be sent from the server to the client; and

a server coupled to the client, wherein the server is sending an HTTP response to the client, wherein the HTTP response has at least a portion of the first portion of the digital media data,

wherein the digital media data comprises a live video data, the live video data representing data to be rendered at the client as an event related to the live video data is being recorded.

25. The system of claim 24 wherein the HTTP POST request comprises a media command for causing the first portion of the digital media data to be sent from the server to the client.

26. A system comprising:

a client sending a Hypertext Transfer Protocol (HTTP protocol) POST request requesting a first portion of a digital media data and including a media command for causing the first portion of the digital media data to be sent from the server to the client; and

a server coupled to the client, wherein the server is sending an HTTP response to the client, wherein the HTTP response has at least a portion of the first portion of the digital media data, wherein the HTTP POST request comprises a media command for causing the first portion of the digital media data to be sent from the server to the client,

wherein the media command includes a REWIND command and wherein the first portion of the digital media data represents digital media data that is earlier in time than a video frame currently displayed at the client.

27. The system of claim 25 wherein the media command includes a FAST FORWARD command and wherein the first portion of the digital media data represents digital media

19

data that is later in time than a video frame currently displayed at the client.

28. The system of claim 25 wherein the media command includes a LIVE PLAY command and wherein the first portion of the digital media data represents digital media data that is recorded live. 5

29. The system of claim 25 wherein the media command includes a REAL-TIME PLAY command and wherein the first portion of the digital media data represents digital media data that is stored earlier and streamed through the server. 10

30. The system of claim 26 wherein a HTTP protocol is permitted on only a selected port of the server, the HTTP protocol request and the HTTP response being multiplexed through the selected port. 15

31. The system of claim 30 wherein the selected port represents one of a port 80 and a port 8080 on the server.

32. The system of claim 31 wherein the server is employed to receive a number of HTTP requests from other clients coupled to the server, the number of HTTP requests also being multiplexed via the selected port. 20

33. The system of claim 32 wherein the number of HTTP requests comprises a unique identification (ID) identifying the digital media data as digital media requested by the client, the unique ID being assigned by the server when the client establishes connection with the server. 25

20

34. A system comprising:

a client sending a Hypertext Transfer Protocol (HTTP protocol) POST request requesting a first portion of a digital media data and including a media command for causing the first portion of the digital media data to be sent from the server to the client; and

a server coupled to the client, wherein the server is sending an HTTP response to the client, wherein the HTTP response has at least a portion of the first portion of the digital media data,

wherein the digital media data comprises video data.

35. A system comprising:

a client sending a Hypertext Transfer Protocol (HTTP protocol) POST request requesting a first portion of a digital media data and including a media command for causing the first portion of the digital media data to be sent from the server to the client; and

a server coupled to the client, wherein the server is sending an HTTP response to the client, wherein the HTTP response has at least a portion of the first portion of the digital media data,

wherein the digital media data comprises audio data.

* * * * *

UNITED STATES PATENT AND TRADEMARK OFFICE
CERTIFICATE OF CORRECTION

PATENT NO. : 6,128,653
DATED : October 3, 2000
INVENTOR(S) : David del Val et al.

Page 1 of 1

It is certified that error appears in the above-identified patent and that said Letters Patent is hereby corrected as shown below:

Title page, Item [54] and Column 1, line 1,

Change "METHOD AND APPARATUS FOR COMMUNICATION MEDIA
COMMANDS AND MEDIA DATA USING THE HTTP PROTOCOL" to
-- METHOD AND APPARATUS FOR COMMUNICATION MEDIA
COMMANDS AND DATA USING THE HTTP PROTOCOL --

Item [22], change "Mar. 17, 1997" to -- Mar. 14, 1997 --

Signed and Sealed this

Tenth Day of September, 2002

Attest:



Attesting Officer

JAMES E. ROGAN
Director of the United States Patent and Trademark Office



US006795848B1

(12) **United States Patent**
Border et al.

(10) **Patent No.:** **US 6,795,848 B1**
(45) **Date of Patent:** **Sep. 21, 2004**

(54) **SYSTEM AND METHOD OF READING
AHEAD OF OBJECTS FOR DELIVERY TO
AN HTTP PROXY SERVER**

(75) Inventors: **John Border**, Germantown, MD (US);
Douglas Dillon, Gaithersburg, MD
(US); **Matt Butehorn**, Mt. Airy, MD
(US)

(73) Assignee: **Hughes Electronics Corporation**, El
Segundo, CA (US)

(*) Notice: Subject to any disclaimer, the term of this
patent is extended or adjusted under 35
U.S.C. 154(b) by 765 days.

(21) Appl. No.: **09/708,134**

(22) Filed: **Nov. 8, 2000**

(51) Int. Cl.⁷ **G06F 15/167**

(52) U.S. Cl. **709/213; 709/203; 709/201**

(58) Field of Search **709/201-203,**
709/238, 225, 226, 229, 213; 711/117,
118, 122, 137

(56) **References Cited**

U.S. PATENT DOCUMENTS

5,995,725 A	*	11/1999	Dillon	709/203
6,112,228 A	*	8/2000	Earl et al.	709/205
6,226,635 B1	*	5/2001	Katariya	707/4
6,282,542 B1	*	8/2001	Carneal et al.	707/10
6,442,651 B2	*	8/2002	Crow et al.	711/118
6,658,463 B1		12/2003	Dillon et al.	709/219

FOREIGN PATENT DOCUMENTS

DE	198 21 876 A	11/1999
WO	WO 98/53410	11/1998
WO	WO 99/08429	2/1999

OTHER PUBLICATIONS

Luotonen A: "Web Proxy Servers", Web Proxy Servers, XX,
XX, 1998 pp. 156-170, XP002928629 p. 170, line 12-p.
170, line 40.

Ari Loutonen, Web Proxy Servers, XP002928629, pp.
156-170, Prentice Hall PTR, Upper Saddle River, NJ.

H. Inoue, et al., "An Adaptive WWW Cache Mechanism in
the AI3 Network", *Proceedings of the INET'97*, www.iso-
c.org/inet97/proceedings/A1/A1_2.HTM, pp. 1-5.

Y. Zhang, et al., "HBX High Bandwidth X for Satellite
Internetworking", 10th Annual X Technical Conference, San
Jose, CA, Feb. 12-14, 1996 (The X Resource, Issue 17, pp.
85-94), pp. 1-10.

T. Baba, et al., "AI³ Satellite Internet Infrastructure and the
Deployment in Asia", IEICE Trans. Commun., vol. E84-B,
No. 8, Aug. 2001, pp. 2048-2057.

H. Inoue, "An Adaptive WWW Cache Mechanism in the
AI3 Network", INET'97 (Jun. 24-27, 1997), www.ai3.net/
pub/inet97/cache_ppt/foils.html, pp. 1-9.

* cited by examiner

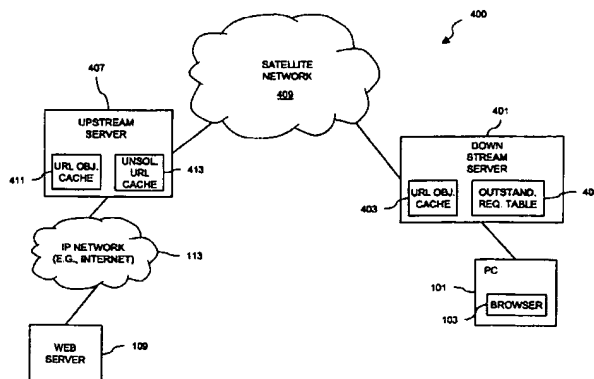
Primary Examiner—Mehmet B. Geckil

(74) *Attorney, Agent, or Firm*—John T. Whelan

(57) **ABSTRACT**

A communication system for retrieving web content is
disclosed. A downstream proxy server receives a URL
request message from a web browser, in which the URL
request message specifies a URL content that has an embed-
ded object. An upstream proxy server receives the URL
request message from the downstream proxy server. The
upstream proxy server selectively forwards the URL request
message to a web server and receives the URL content from
the web server, wherein the upstream proxy server forwards
the URL content to the downstream proxy server and parses
the URL content to obtain the embedded object prior to
receiving a corresponding embedded object request message
initiated by the web browser.

4 Claims, 7 Drawing Sheets



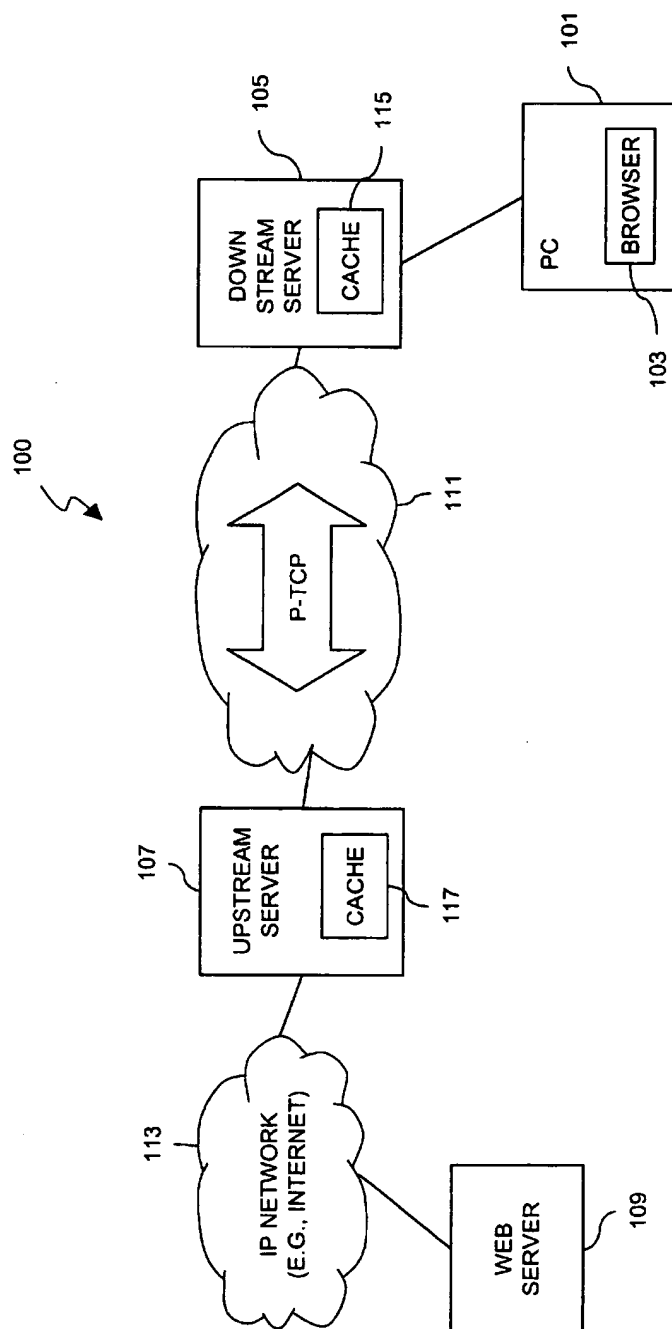


FIG. 1

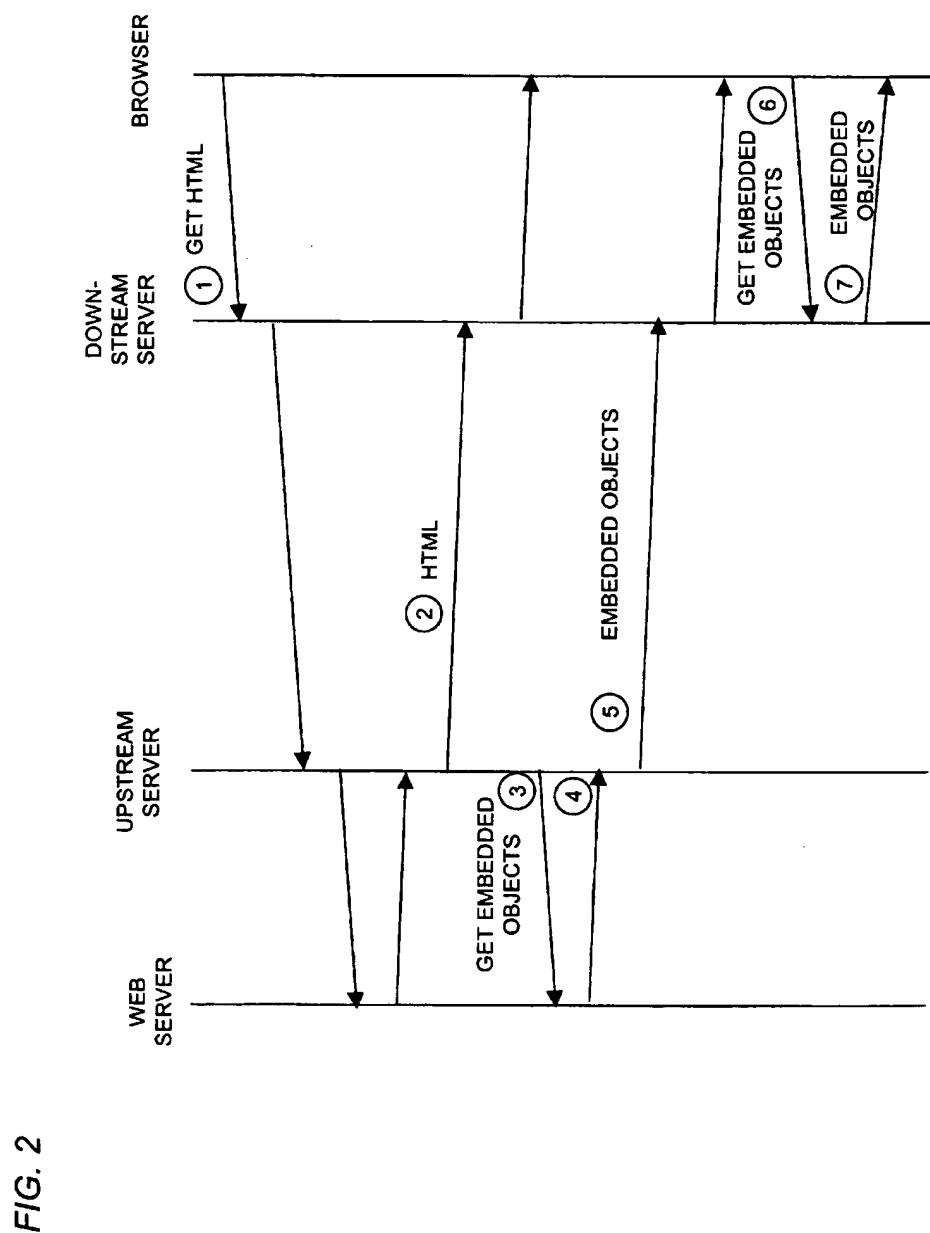
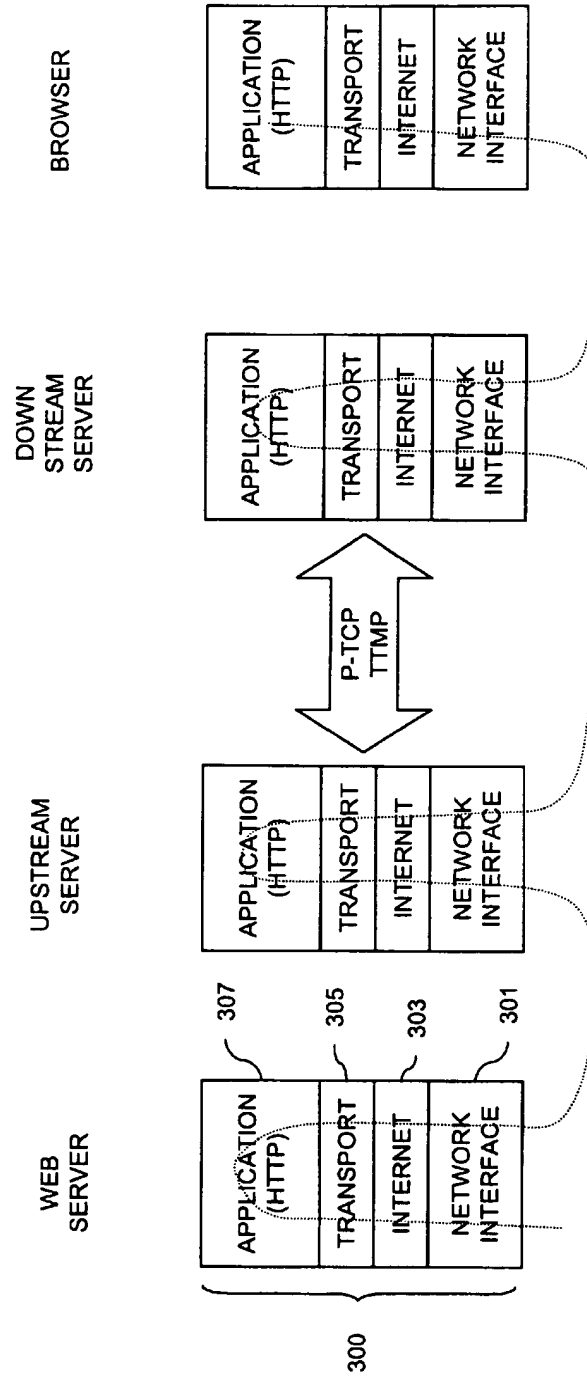
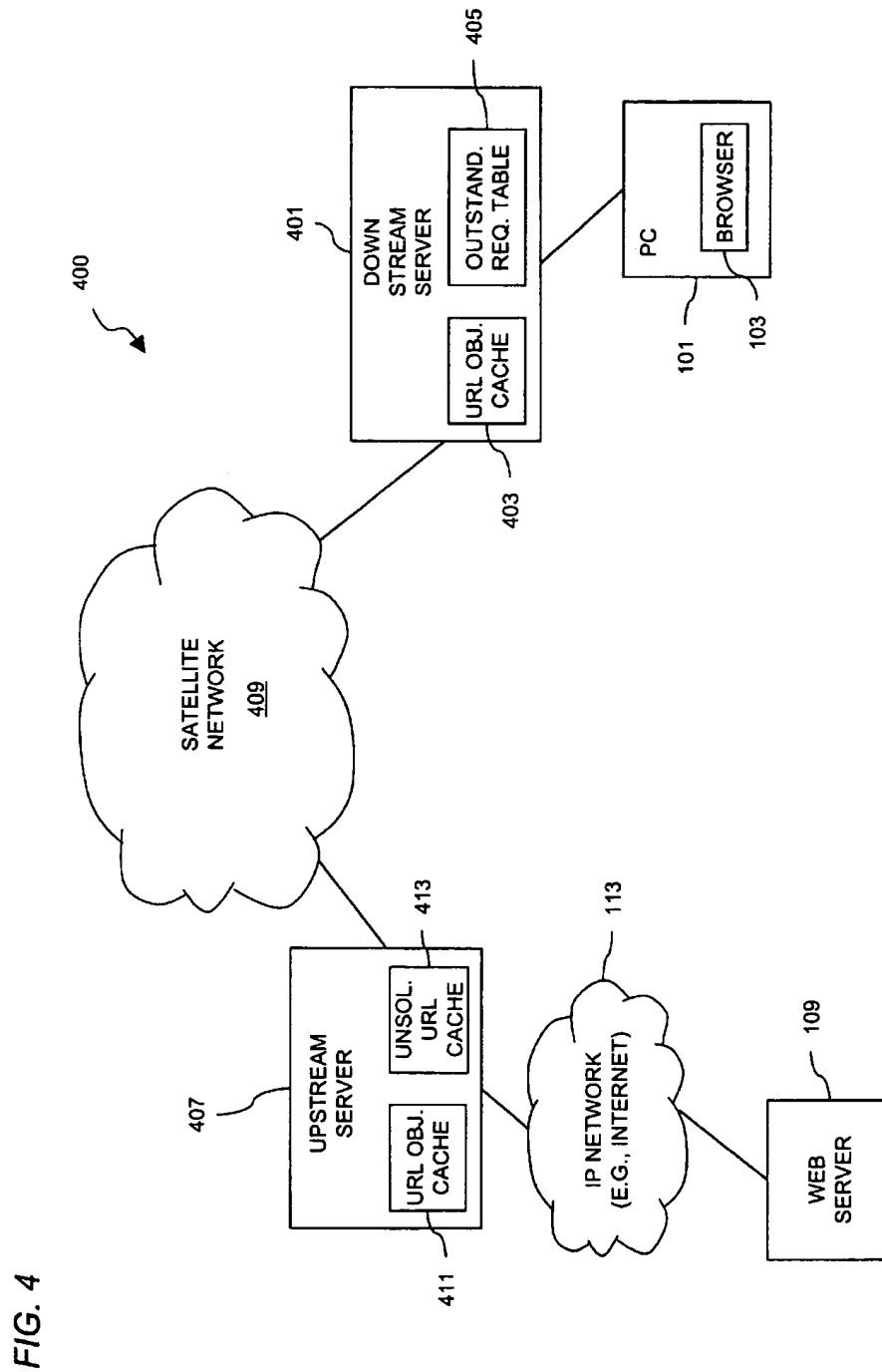


FIG. 3





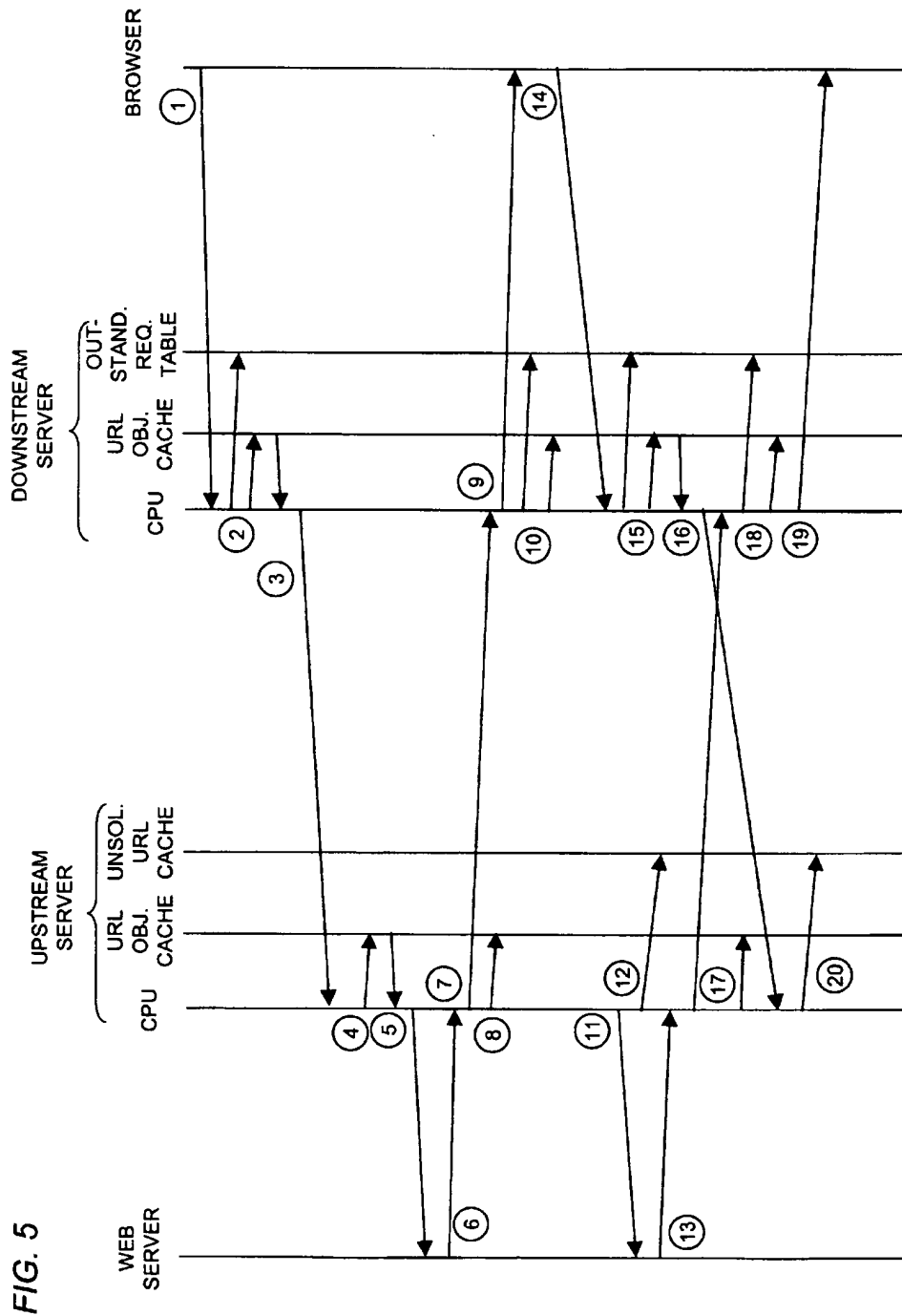


FIG. 6

PRIOR ART

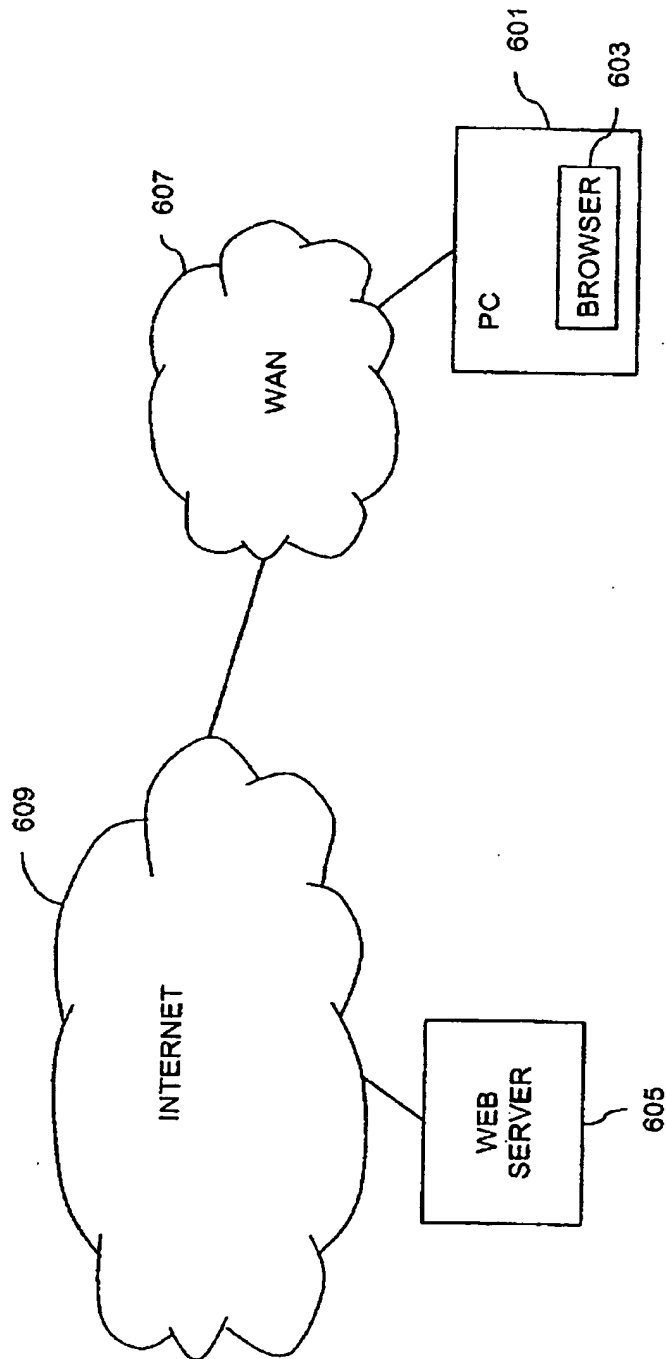
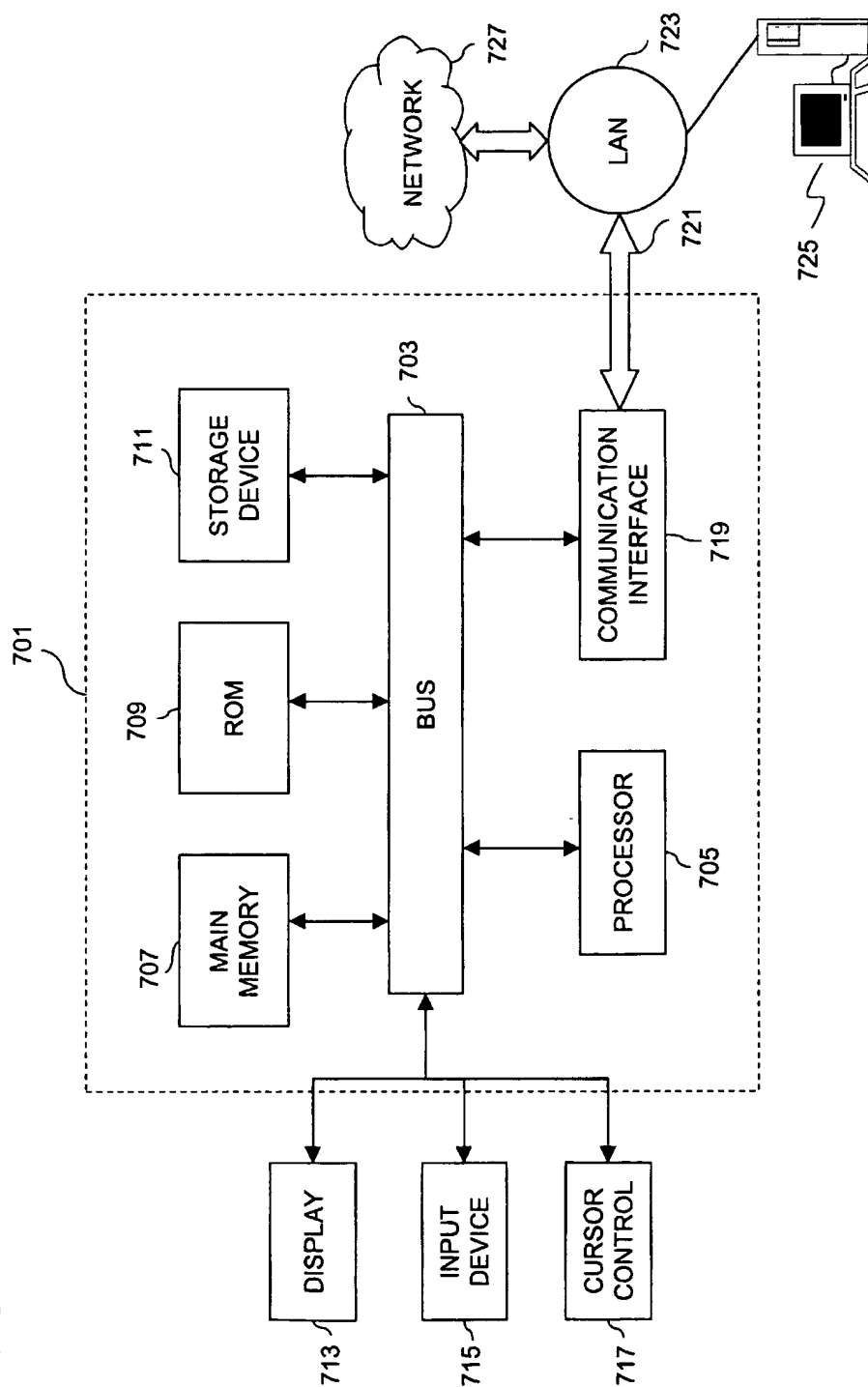


FIG. 7



1

SYSTEM AND METHOD OF READING AHEAD OF OBJECTS FOR DELIVERY TO AN HTTP PROXY SERVER

CROSS-REFERENCES TO RELATED APPLICATION

This application is related to co-pending U.S. patent application Ser. No. 09/498,936, filed Feb. 4, 2000, entitled "Satellite Multicast Performance Enhancing Multicast HTTP Proxy System and Method," the entirety of which is incorporated herein by reference.

BACKGROUND OF THE INVENTION

1. Field of the Invention

The present invention relates to a communication system, and is more particularly related to retrieving web content using proxy servers.

2. Discussion of the Background

As businesses and society, in general, become increasingly reliant on communication networks to conduct a variety of activities, ranging from business transactions to personal entertainment, these communication networks continue to experience greater and greater delay, stemming in part from traffic congestion and network latency. For example, the maturity of electronic commerce and acceptance of the Internet, in particular the World Wide Web ("Web"), as a daily tool pose an enormous challenge to communication engineers to develop techniques to reduce network latency and user response times. With the advances in processing power of desktop computers, the average user has grown accustomed to sophisticated applications (e.g., streaming video, radio broadcasts, video games, etc.), which place tremendous strain on network resources. The Web as well as other Internet services rely on protocols and networking architectures that offer great flexibility and robustness; however, such infrastructure may be inefficient in transporting Web traffic, which can result in large user response time, particularly if the traffic has to traverse an intermediary network with a relatively large latency (e.g., a satellite network).

FIG. 6 is a diagram of a conventional communication system for providing retrieval of web content by a personal computer (PC). PC 601 is loaded with a web browser 603 to access the web pages that are resident on web server 605; collectively the web pages and web server 605 denote a "web site." PC 603 connects to a wide area network (WAN) 607, which is linked to the Internet 609. The above arrangement is typical of a business environment, whereby the PC 601 is networked to the Internet 609. A residential user, in contrast, normally has a dial-up connection (not shown) to the Internet 609 for access to the Web. The phenomenal growth of the Web is attributable to the ease and standardized manner of "creating" a web page, which can possess textual, audio, and video content.

Web pages are formatted according to the Hypertext Markup Language (HTML) standard which provides for the display of high-quality text (including control over the location, size, color and font for the text), the display of graphics within the page and the "linking" from one page to another, possibly stored on a different web server. Each HTML document, graphic image, video clip or other individual piece of content is identified, that is, addressed, by an Internet address, referred to as a Uniform Resource Locator (URL). As used herein, a "URL" may refer to an address of an individual piece of web content (HTML document,

2

image, sound-clip, video-clip, etc.) or the individual piece of content addressed by the URL. When a distinction is required, the term "URL address" refers to the URL itself while the terms "web content", "URL content" or "URL object" refers to the content addressed by the URL.

In a typical transaction, the user enters or specifies a URL to the web browser 603, which in turn requests a URL from the web server 605. The web server 605 returns an HTML page, which contains numerous embedded objects (i.e., web content), to the web browser 603. Upon receiving the HTML page, the web browser 603 parses the page to retrieve each embedded object. The retrieval process often requires the establishment of separate communication sessions (e.g., TCP (Transmission Control Protocol) sessions) to the web server 605. That is, after an embedded object is received, the TCP session is torn down and another TCP session is established for the next object. Given the richness of the content of web pages, it is not uncommon for a web page to possess over 30 embedded objects. This arrangement disadvantageously consumes network resources, but more significantly, introduces delay to the user.

Delay is further increased if the WAN 607 is a satellite network, as the network latency of the satellite network is conventionally a longer latency than terrestrial networks. In addition, because HTTP utilizes a separate TCP connection for each transaction, the large number of transactions amplifies the network latency. Further, the manner in which frames are created and images are embedded in HTML requires a separate HTTP transaction for every frame and URL compounds the delay.

Based on the foregoing, there is a clear need for improved approaches for retrieval of web content within a communication system.

There is a need to utilize standard protocols to avoid development costs and provide rapid industry acceptance.

There is also a need for a web content retrieval mechanism that makes the networks with relatively large latency viable and/or competitive for Internet access.

Therefore, an approach for retrieving web content that reduces user response times is highly desirable.

SUMMARY OF THE INVENTION

According to one aspect of the invention, a communication system for retrieving web content comprises a downstream proxy server that is configured to receive a URL request message from a web browser. The URL request message specifies a URL content that has an embedded object. An upstream proxy server is configured to communicate with the downstream proxy server and to receive the URL request message from the downstream proxy server. The upstream proxy server selectively forwards the URL request message to a web server and receives the URL content from the web server, wherein the upstream proxy server forwards the URL content to the downstream proxy server and parses the URL content to obtain the embedded object prior to the web browser having to issue an embedded object request message. The above arrangement advantageously reduces user response time associated with web browsing.

BRIEF DESCRIPTION OF THE DRAWINGS

A more complete appreciation of the invention and many of the attendant advantages thereof will be readily obtained as the same becomes better understood by reference to the following detailed description when considered in connection with the accompanying drawings, wherein:

3

FIG. 1 is a diagram of a communication system employing a downstream proxy server and an upstream proxy server for accessing a web server, according to an embodiment of the present invention;

FIG. 2 is a sequence diagram of the process of reading ahead used in the system of FIG. 1;

FIG. 3 is a block diagram of the protocols utilized in the system of FIG. 1;

FIG. 4 is a diagram of a communication system employing a downstream proxy server and an upstream proxy server that maintains an unsolicited URL (Uniform Resource Locator) cache for accessing a web server, according to an embodiment of the present invention;

FIG. 5 is a sequence diagram of the process of reading ahead used in the system of FIG. 4;

FIG. 6 is a diagram of a conventional communication system for providing retrieval of web content by a personal computer (PC); and

FIG. 7 is a diagram of a computer system that can be configured as a proxy server, in accordance with an embodiment of the present invention.

DESCRIPTION OF THE PREFERRED EMBODIMENTS

In the following description, for the purpose of explanation, specific details are set forth in order to provide a thorough understanding of the invention. However, it will be apparent that the invention may be practiced without these specific details. In some instances, well-known structures and devices are depicted in block diagram form in order to avoid unnecessarily obscuring the invention.

The present invention provides a communication system for retrieving web content. A downstream proxy server receives a URL request message from a web browser, in which the URL request message specifies a URL content that has an embedded object. An upstream proxy server receives the URL request message from the downstream proxy server. The upstream proxy server selectively forwards the URL request message to a web server and receives the URL content from the web server. The upstream proxy server forwards the URL content to the downstream proxy server and parses the URL content to obtain the embedded object prior to receiving a corresponding embedded object request message initiated by the web browser.

Although the present invention is discussed with respect to a protocols and interfaces to support communication with the Internet, the present invention has applicability to any protocols and interfaces to support a packet switched network, in general.

FIG. 1 shows a diagram of a communication system employing a downstream proxy server and an upstream proxy server for accessing a web server, according to an embodiment of the present invention. Communication system 100 includes a user station 101 that utilizes a standard web browser 103 (e.g., Microsoft Internet Explorer, Netscape Navigator). In this example, the user station 101 is a personal computer (PC); however, any computing platform may be utilized, such as a workstation, web enabled set-top boxes, web appliances, etc. System 100 utilizes two proxy servers 105 and 107, which are referred to as a downstream proxy server 105 and an upstream proxy server 107, respectively. PC 101 connects to downstream server 105, which communicates with upstream server 107 through a network 111. This communication with downstream server 105 may be transparent to PC 101. According to an embodiment of

4

the present invention, the network 111 is a VSAT (Very Small Aperture Terminal) satellite network. Alternatively, the network 111 may be any type of Wide Area Network (WAN); e.g., ATM (Asynchronous Transfer Mode) network, router-based network, T1 network, etc. The upstream server 107 has connectivity to an IP network 113, such as the Internet, to access web server 109.

Proxy servers 105 and 107, according to an embodiment of the present invention, are Hypertext Transfer Protocol (HTTP) proxy servers with HTTP caches 115 and 117, respectively. These servers 105 and 107 communicate using persistent connections (which is a feature of HTTP 1.1). Use of persistent connections enables a single TCP connection to be reused for multiple requests of the embedded objects within a web page associated with web server 109. Further, TCP Transaction Multiplexing Protocol (TTMP) may be utilized. TTMP and persistent-TCP are more fully described with respect to FIG. 3.

Web browser 103 may be configured to either access URLs directly from a web server 109 or from HTTP proxy servers 105 and 107. A web page may refer to various source documents by indicating the associated URLs. As discussed above, a URL specifies an address of an "object" in the Internet 113 by explicitly indicating the method of accessing the resource. A representative format of a URL is as follows: <http://www.hns.com/homepage/document.html>. This example indicates that the file "document.html" is accessed using HTTP.

HTTP proxy server 105 and 107 acts as an intermediary between one or more browsers and many web servers (e.g., server 109). A web browser 103 requests a URL from the proxy server (e.g., 105) which in turn "gets" the URL from the addressed web server 109. An HTTP proxy 105 itself may be configured to either access URLs directly from a web server 109 or from another HTTP proxy server 107.

According to one embodiment of the present invention, the proxy servers 105 and 107 may support multicast delivery. IP multicasting can be used to transmit information from upstream server 107 to multiple downstream servers (of which only one downstream server 105 is shown). A multicast receiver (e.g., a network interface card (NIC)) for the downstream proxy server 105 operates in one of two modes: active and inactive. In the active mode operation, the downstream proxy server 105 opens multicast addresses and actively processes the received URLs on those addresses. During the inactive mode, the downstream proxy server 105 disables multicast reception from the upstream proxy server 107. In the inactive state the downstream proxy server 105 minimizes its use of resources by, for example, closing the cache and freeing its RAM memory (not shown).

For downstream proxy server 105 operating on a general purpose personal computer, the multicast receiver for the downstream proxy server 105 may be configured to switch between the active and inactive states to minimize the proxy server's interfering with user-directed processing. The downstream proxy server 105 utilizes an activity monitor which monitors user input (key clicks and mouse clicks) to determine when it should reduce resource utilization. The downstream proxy server 105 also monitors for proxy cache lookups to determine when it should go active.

Upon boot up, the multicast receiver is inactive. After a certain amount of time with no user interaction and no proxy cache lookups (e.g., 10 minutes), the downstream proxy server 105 sets the multicast receiver active. The downstream proxy server 105 sets the multicast receiver active immediately upon needing to perform a cache lookup. The

5

downstream proxy server 105 sets the multicast receiver inactive whenever user activity is detected and the cache 115 has not had any lookups for a configurable period of time (e.g., 5 minutes).

For downstream proxy servers 105 running on systems with adequate CPU (central processing unit) resources to simultaneously handle URL reception and other applications, the user may configure the downstream proxy server 105 to set the multicast receiver to stay active regardless of user activity. The operation of system 100 in the retrieval of web content, according to an embodiment of the present invention, is described in FIG. 2, below.

FIG. 2 shows a sequence diagram of the process of reading ahead used in the system of FIG. 1. To retrieve a web page (i.e., HTML page) from web server 109, the web browser 103 on PC 101 issues an HTTP GET request, which is received by downstream proxy server 105 (per step 1). For the purposes of explanation, the HTML page is addressed as URL "HTML." The downstream server 105 checks its cache 115 to determine whether the requested URL has been previously visited. If the downstream proxy server 105 does not have URL HTML stored in cache 115, the server 105 relays this request, GET URL "HTML", to upstream server 117.

The HTTP protocol also supports a GET IF MODIFIED SINCE request wherein a web server (or a proxy server) either responds with a status code indicating that the URL has not changed or with the URL content if the URL has changed since the requested date and time. This mechanism updates cache 115 of proxy server 105 only if the contents have changed, thereby saving unnecessary processing costs.

The upstream server 117 in turn searches for the URL HTML in its cache 117; if the HTML page is not found in cache 117, the server 117 issues the GET URL HTML request to the web server 109 for the HTML page. Next, in step 2, the web server 109 transmits the requested HTML page to the upstream server 117, which stores the received HTML page in cache 117. The upstream server 117 forwards the HTML page to the downstream server 105, and ultimately to the web browser 103. The HTML page is stored in cache 115 of the downstream server 105 as well as the web browser's cache (not shown). In step 3, the upstream server 117 parses the HTML page and requests the embedded objects within the HTML page from the web server 109; the embedded objects are requested prior to receiving corresponding embedded object requests initiated by the web browser 103.

Step 3 may involve the issuance of multiple GET requests; the web page within web server 109 may contain over 30 embedded objects, thus requiring 30 GET requests. In effect, this scheme provides a way to "read ahead" (i.e., retrieve the embedded object) in anticipation of corresponding requests by the web browser 103. The determination to read-ahead may be based upon explicit tracking of the content of the downstream server cache 115; only those embedded objects that are not found in the cache 115 are requested. Alternatively, the upstream server 107 may only request those embedded objects that are not in the upstream server cache 117. Further, in actual implementation wherein multiple web servers exist, the upstream server 107 may track which web server tend to transmit uncacheable objects; for such servers, objects stored therein are read-ahead.

Moreover if the HTML contains a cookie and the GET HTML request is directed to the same web server, then the upstream server 107 includes the cookie in the read-ahead request to the web server 109 for the embedded objects. A

6

cookie is information that a web server 109 stores on the client system, e.g., PC 101, to identify the client system. Cookies provide a way for the web server 109 to return customized web pages to the PC 101.

In step 4, the web server 109 honors the GET request by transmitting the embedded objects to the upstream server 107. The upstream server 107, as in step 5, then forwards the retrieved objects to the downstream server 105, where the objects are stored until they are requested by the web browser 103. It should be noted that the upstream server 107 forwards the embedded objects prior to being requested to do so by the web browser 103; however, the upstream server 107 performs this forwarding step based on an established criteria. There are scenarios in which all the embedded objects that are read-ahead may not subsequently be requested by the web browser 103. In such cases, if the upstream server 107 transfers these embedded objects over network 111 to the downstream server 105, the bandwidth of network 111 would be wasted, along with the resources of the downstream server 105. Accordingly, the forwarding criteria need to reflect the trade off between response time and bandwidth utilization. These forwarding criteria may include the following: (1) object size, and (2) "cachability." That is, upstream server 107 may only forward objects that are of a predetermined size or less, so that large objects (which occupy greater bandwidth) are not sent to the downstream server 105. Additionally, if the embedded object is marked uncacheable, then the object may be forwarded to the downstream server 105, which by definition will not have the object stored. The upstream server 107 may be configured to forward every retrieved embedded object, if bandwidth is not a major concern.

In the scenario in which the embedded objects correspond to a request that contains a cookie, the upstream server 107 provides an indication whether the embedded objects has the corresponding cookie.

In step 6, the web browser 103 issues a GET request for the embedded objects corresponding to the web page within the web server 109. The downstream server 105 recognizes that the requested embedded objects are stored within its cache 115 and forwards the embedded objects to the web browser 103. Under this approach, the delays associated with network 111 and the Internet 113 are advantageously avoided.

The caching HTTP proxy servers 105 and 107, according to one embodiment of the present invention, stores the most frequently accessed URLs. When web server 109 delivers a URL to the proxy servers 105 and 107, the web server 109 may deliver along with the URL an indication of whether the URL should not be cached and an indication of when the URL was last modified.

At this point, web browser 103 has already requested URL HTML, and has the URL HTML stored in a cache (not shown) of the PC 101. To avoid stale information, the web browser 103 needs to determine whether the information stored at URL HTML has been updated since the time it was last requested. As a result, the browser 103 issues a GET HTML IF MODIFIED SINCE the last time HTML was obtained. Assuming that URL HTML was obtained at 11:30 a.m. on Sep. 22, 2000, browser 103 issues a GET HTML IF MODIFIED SINCE Sep. 22, 2000 at 11:30 a.m. request. This request is sent to downstream proxy server 105. If downstream proxy server 105 has received an updated version of URL HTML since Sep. 22, 2000 at 11:30 a.m., downstream proxy server 105 supplies the new URL HTML information to the browser 103. If not, the downstream

proxy server 105 issues a GET IF MODIFIED SINCE command to upstream proxy server 107. If upstream proxy server 107 has received an updated URL HTML since Sep. 22, 2000 at 11:30 a.m., upstream proxy server 107 passes the new URL HTML to the downstream proxy server 105. If not, the upstream proxy server 107 issues a GET HTML IF MODIFIED SINCE command to the web server 109. If URL HTML has not changed since Sep. 22, 2000 at 11:30 a.m., web server 109 issues a NO CHANGE response to the upstream proxy server 107. Under this arrangement, bandwidth and processing time are saved, since if the URL HTML has not been modified since the last request, the entire contents of URL HTML need not be transferred between web browser 103, downstream proxy server 105, upstream proxy server 107, and the web server 109, only an indication that there has been no change need be exchanged. Caching proxy servers 105 and 107 offer both reduced network utilization and reduced response time when they are able to satisfy requests with cached URLs.

FIG. 3 shows a block diagram of the protocols utilized in the system of FIG. 1. The servers 105, 107, and 109 and PC 101 employ, according to one embodiment of the present invention, a layered protocol stack 300. The protocol stack 300 includes a network interface layer 301, an Internet layer 303, a transport layer 305, and an application layer 307.

HTTP is an application level protocol that is employed for information transfer over the Web. RFC (Request for Comment) 2616 specifies this protocol and is incorporated herein in its entirety. In addition, a more detailed definition of URL can be found in RFC 1737, which is incorporated herein in its entirety.

The Internet layer 303 may be the Internet Protocol (IP) version 4 or 6, for instance. The transport layer 305 may include the TCP (Transmission Control Protocol) and the UDP (User Datagram Protocol). According to one embodiment of the present invention, at the transport layer, persistent TCP connections are utilized in the system 100; in addition, TCP Transaction Multiplexing Protocol (TTMP) may be used.

The TCP Transaction Multiplexing Protocol (TTMP) allows multiple transactions, in this case HTTP transactions, to be multiplexed onto one TCP connection. Thus, transaction multiplexing provides an improvement over separate connection for each transaction (HTTP 1.0) and pipelining (HTTP 1.1) by preventing a single stalled request from stalling other requests. This is particularly beneficial when the downstream proxy server 105 is supporting simultaneous requests from multiple browsers (of which only browser 103 is shown in FIG. 1).

The downstream proxy server 105 initiates and maintains a TCP connection to the upstream proxy server 107 as needed to carry HTTP transactions. The TCP connection could be set up and kept connected as long as the downstream proxy server 105 is running and connected to the network 111. The persistent TCP connection may also be set up when the first transaction is required and torn down after the connection has been idle for some period.

An HTTP transaction begins with a request header, optionally followed by request content which is sent from the downstream proxy server 105 to the upstream proxy server 107. An HTTP transaction concludes with a response header, optionally followed by response content. The downstream proxy server 105 maintains a transaction ID sequence number, which is incremented with each transaction. The downstream proxy server 105 breaks the transaction request into one or more blocks, creates a TTMP header for each

block, and sends the blocks with a TTMP header to the upstream proxy server 107. The upstream proxy server 107 similarly breaks a transaction response into blocks and sends the blocks with a TTMP header to the downstream proxy server 105. The TTMP header contains the information necessary for the upstream proxy server 107 to reassemble a complete transaction command and to return the matching transaction response.

In particular, the TTMP header contains the following fields: a transaction ID field, a Block Length field, a Last Indication field, an Abort Indication field, and a Compression Information field. The transaction ID (i.e., the transaction sequence number) must rollover less frequently than the maximum number of supported outstanding transactions. The Block Length field allows a proxy server 105 and 107 to determine the beginning and ending of each block. The Last Indication field allows the proxy server 105 and 107 to determine when the end of a transaction response has been received. The Abort Indication field allows the proxy server 105 and 107 to abort a transaction when the transaction request or response cannot be completed. Lastly, the Compression Information field defines how to decompress the block.

The use of a single HTTP connection reduces the number of TCP acknowledgements that are sent over the network 111. Reduction in the number of TCP acknowledgements significantly reduces the use of inbound networking resources which is particularly important when the network 111 is a VSAT system or other wireless systems. This reduction of acknowledgements is more significant when techniques, such as those described in U.S. Pat. No. 5,995,725 to Dillon entitled "Method and Apparatus for Requesting and Retrieving Information for a Source Computer Using Terrestrial and Satellite Interface" issued Nov. 30, 1999 (which is incorporated herein in its entirety), minimize the number of TCP acknowledgements per second per TCP connection.

Alternatively, downstream proxy server 105, for efficiency, may use the User Datagram Protocol (UDP) to transmit HTTP GET and GET IF MODIFIED SINCE requests to the upstream proxy server 107. This is done by placing the HTTP request header into the UDP payload. The use of UDP is very efficient as the overhead of establishing, maintaining and clearing TCP connections is not incurred. It is "best effort" in that there is no guarantee that the UDP packets will be delivered.

FIG. 4 shows a diagram of a communication system employing a downstream proxy server and an upstream proxy server that maintains an unsolicited URL (Uniform Resource Locator) cache for accessing a web server, according to an embodiment of the present invention. Communication system 400 employs a downstream server 401 that utilizes a cache 403 to store URL objects (i.e., embedded objects) as well as an Outstanding Request table 405. The table 405 tracks the URL requests that the downstream server 401 has forwarded to upstream server 407. In an embodiment of the present invention, the downstream server 401 and the upstream server 407 communicate over a satellite network 409. The upstream server 407 maintains a URL object cache 411 for storing the embedded objects that are retrieved from web server 109. In addition, the upstream server 407 uses an unsolicited URL cache 413, which stores the URL requests for embedded objects in advance of the web browser 103 initiating such requests. The above arrangement advantageously enhances system performance.

FIG. 5 is a sequence diagram of the process of reading ahead used in the system of FIG. 4. In step 1, the web

browser 101 sends a GET request (e.g., GET x.html) to the downstream server 401. The downstream server 401 checks the URL object cache 403 to determine whether x.html is stored in cache 403; if the content is stored in cache 403, the downstream server 401 forwards the content to the browser 103. Otherwise, the downstream server 401 writes the request in the Outstanding Request table 405 and sends the GET request to the upstream server 407 (step 3). In this case, the web browser 103 and the downstream server 401 have not encountered the requested html page before. However, in the event that the web browser 103 has requested this HTML in the past or the downstream server 401 has stored this HTML previously, the latest time stamp is passed to the upstream server as a conditional GET request (e.g., GET IF MODIFIED SINCE Sep. 22, 2000). In this manner, only content that is more updated than the time stamp are retrieved. In step 4, the upstream server 407 checks the URL object cache 411 in response to the received GET x.html request. Assuming x.html is not found in the URL object cache 411, the upstream server 407 forwards the GET x.html request to the web server 109, per step 5. Accordingly, the web server 109, as in step 6, returns the web page to the upstream server 407. In turn, the upstream server 407 forwards the web page to the downstream server 401, as in step 7, and stores the web page in the URL object cache 411, per step 8. In step 9, the downstream server 401 sends the received web page to the web browser 103. At this time, the downstream server 401 deletes the corresponding entry in the Outstanding Request table 405 and stores the received web page in the URL object cache 411 (step 10).

Concurrent with steps 9 and 10, the upstream server 407 parses the web page. The upstream server 407 then makes a determination, as in step 11, to read-ahead the embedded objects of the web page based upon the read-ahead criteria that were discussed with respect to FIG. 2, using a series of GET embedded object requests. Consequently, the upstream server 407 stores the URL x.html in the unsolicited URL cache 413, per step 12. In step 13, the web server 109 returns the embedded objects to the upstream server 407.

In step 14, the web browser 103 parses the x.html page and issues a series of GET embedded objects requests. However, for explanatory purposes, FIG. 5 shows a single transaction for step 14. In step 15, the downstream server 401 checks its URL object cache 403 for the requested embedded object, and, assuming the particular object is not stored in cache 403, writes an entry in the Outstanding Request table 405 corresponding to the GET embedded object request. Next, the downstream server 401 forwards the GET embedded object request to the upstream server 407 (per step 16).

As shown in FIG. 5, prior to the upstream server 407 receiving the GET embedded object request from the downstream server 401, the upstream server 407 forwards the embedded objects to the downstream server 401 based on a forwarding criteria (as previously discussed with respect to FIG. 2), storing these embedded objects in the URL object cache 413. In step 18, the downstream server 401 updates the Outstanding Request table 405 by deleting the GET embedded object request from the web browser 103, and stores the received embedded object that has been read-ahead from the upstream server 407. The embedded object is then transferred to the web browser 103 (per step 19). In step 20, upon receiving the GET embedded object from the downstream server 401, the upstream server 407 discards the corresponding URL in the Unsolicited URL cache 413. Under the above approach, the effects of network latencies

associated with satellite network 409 and the Internet 113 are minimized, in that the web browser 103 receives the requested embedded object without having to wait for the full processing and transmission time associated with its GET embedded object request.

FIG. 7 is a diagram of a computer system that can be configured as a proxy server, in accordance with an embodiment of the present invention. Computer system 701 includes a bus 703 or other communication mechanism for communicating information, and a processor 705 coupled with bus 703 for processing the information. Computer system 701 also includes a main memory 707, such as a random access memory (RAM) or other dynamic storage device, coupled to bus 703 for storing information and instructions to be executed by processor 705. In addition, main memory 707 may be used for storing temporary variables or other intermediate information during execution of instructions to be executed by processor 705. Computer system 701 further includes a read only memory (ROM) 709 or other static storage device coupled to bus 703 for storing static information and instructions for processor 705. A storage device 711, such as a magnetic disk or optical disk, is provided and coupled to bus 703 for storing information and instructions.

Computer system 701 may be coupled via bus 703 to a display 713, such as a cathode ray tube (CRT), for displaying information to a computer user. An input device 715, including alphanumeric and other keys, is coupled to bus 703 for communicating information and command selections to processor 705. Another type of user input device is cursor control 717, such as a mouse, a trackball, or cursor direction keys for communicating direction information and command selections to processor 705 and for controlling cursor movement on display 713.

According to one embodiment, interaction within system 100 is provided by computer system 701 in response to processor 705 executing one or more sequences of one or more instructions contained in main memory 707. Such instructions may be read into main memory 707 from another computer-readable medium, such as storage device 711. Execution of the sequences of instructions contained in main memory 707 causes processor 705 to perform the process steps described herein. One or more processors in a multi-processing arrangement may also be employed to execute the sequences of instructions contained in main memory 707. In alternative embodiments, hard-wired circuitry may be used in place of or in combination with software instructions. Thus, embodiments are not limited to any specific combination of hardware circuitry and software.

Further, the instructions to support the system interfaces and protocols of system 100 may reside on a computer-readable medium. The term "computer-readable medium" as used herein refers to any medium that participates in providing instructions to processor 705 for execution. Such a medium may take many forms, including but not limited to, non-volatile media, volatile media, and transmission media. Non-volatile media includes, for example, optical or magnetic disks, such as storage device 711. Volatile media includes dynamic memory, such as main memory 707. Transmission media includes coaxial cables, copper wire and fiber optics, including the wires that comprise bus 703. Transmission media can also take the form of acoustic or light waves, such as those generated during radio wave and infrared data communication.

Common forms of computer-readable media include, for example, a floppy disk, a flexible disk, hard disk, magnetic

tape, or any other magnetic medium, a CD-ROM, any other optical medium, punch cards, paper tape, any other physical medium with patterns of holes, a RAM, a PROM, and EPROM, a FLASH-EPROM, any other memory chip or cartridge, a carrier wave as described hereinafter, or any other medium from which a computer can read.

Various forms of computer readable media may be involved in carrying one or more sequences of one or more instructions to processor 705 for execution. For example, the instructions may initially be carried on a magnetic disk of a remote computer. The remote computer can load the instructions relating to the issuance of read-ahead requests remotely into its dynamic memory and send the instructions over a telephone line using a modem. A modem local to computer system 701 can receive the data on the telephone line and use an infrared transmitter to convert the data to an infrared signal. An infrared detector coupled to bus 703 can receive the data carried in the infrared signal and place the data on bus 703. Bus 703 carries the data to main memory 707, from which processor 705 retrieves and executes the instructions. The instructions received by main memory 707 may optionally be stored on storage device 711 either before or after execution by processor 705.

Computer system 701 also includes a communication interface 719 coupled to bus 703. Communication interface 719 provides a two-way data communication coupling to a network link 721 that is connected to a local network 723. For example, communication interface 719 may be a network interface card to attach to any packet switched local area network (LAN). As another example, communication interface 719 may be an asymmetrical digital subscriber line (ADSL) card, an integrated services digital network (ISDN) card or a modem to provide a data communication connection to a corresponding type of telephone line. Wireless links may also be implemented. In any such implementation, communication interface 719 sends and receives electrical, electromagnetic or optical signals that carry digital data streams representing various types of information.

Network link 721 typically provides data communication through one or more networks to other data devices. For example, network link 721 may provide a connection through local network 723 to a host computer 725 or to data equipment operated by a service provider, which provides data communication services through a communication network 727 (e.g., the Internet). LAN 723 and network 727 both use electrical, electromagnetic or optical signals that carry digital data streams. The signals through the various networks and the signals on network link 721 and through communication interface 719, which carry the digital data to and from computer system 701, are exemplary forms of carrier waves transporting the information. Computer system 701 can transmit notifications and receive data, including program code, through the network(s), network link 721 and communication interface 719.

The techniques described herein provide several advantages over prior approaches to retrieving web pages. A downstream proxy server is configured to receive a URL request message from a web browser, wherein the URL request message specifies a URL content that has an embedded object. An upstream proxy server is configured to communicate with the downstream proxy server and to receive the URL request message from the downstream proxy server. The upstream proxy server selectively forwards the URL request message to a web server and receives the URL content from the web server, wherein the upstream proxy server forwards the URL content to the downstream proxy server and parses the URL content to obtain the

embedded object prior to receiving a corresponding embedded object request message initiated by the web browser. This approach advantageously improves user response time.

Obviously, numerous modifications and variations of the present invention are possible in light of the above teachings. It is therefore to be understood that within the scope of the appended claims, the invention may be practiced otherwise than as specifically described herein.

What is claimed is:

1. A communication system for retrieving web content, comprising:

a downstream proxy server configured to receive a URL request message from a web browser, the URL request message specifying a URL content having an embedded object; and

an upstream proxy server configured to communicate with the downstream proxy server and to receive the URL request message from the downstream proxy server, the upstream proxy server selectively forwarding the URL request message to a web server and receiving the URL content from the web server, wherein the upstream proxy server forwards the URL content to the downstream proxy server and parses the URL content to obtain the embedded object prior to receiving a corresponding embedded object request message initiated by the web browser,

wherein the upstream proxy server decides whether or not to send the embedded object to the downstream proxy server in accordance with the size of the embedded object.

2. A communication system for retrieving web content, comprising:

a downstream proxy server configured to receive a URL request message from a web browser, the URL request message specifying a URL content having an embedded object; and

an upstream proxy server configured to communicate with the downstream proxy server and to receive the URL request message from the downstream proxy server, the upstream proxy server selectively forwarding the URL request message to a web server and receiving the URL content from the web server, wherein the upstream proxy server forwards the URL content to the downstream proxy server and parses the URL content to obtain the embedded object prior to receiving a corresponding embedded object request message initiated by the web browser,

wherein the upstream proxy server decides whether or not to send the embedded object to the downstream proxy server in accordance with cachability of the embedded object.

3. A communication system for retrieving web content, comprising:

a downstream proxy server configured to receive a URL request message from a web browser, the URL request message specifying a URL content having an embedded object; and

an upstream proxy server configured to communicate with the downstream proxy server and to receive the URL request message from the downstream proxy server, the upstream proxy server selectively forwarding the URL request message to a web server and receiving the URL content from the web server, wherein the upstream proxy server forwards the URL content to the downstream proxy server and parses the URL content to obtain the embedded object prior to receiving a corre-

13

sponding embedded object request message initiated by the web browser,
 wherein in the case that the URL content has a cookie, the upstream proxy server uses the cookie when obtaining the embedded object.
 5
 4. A communication system for retrieving web content, comprising:
 a downstream proxy server configured to receive a URL request message from a web browser, the URL request message specifying a URL content having an embedded object;
 10
 an upstream proxy server configured to communicate with the downstream proxy server and to receive the URL

14

request message from the downstream proxy server, the upstream proxy server selectively forwarding the URL request message to a web server and receiving the URL content from the web server, wherein the upstream proxy server forwards the URL content to the downstream proxy server and parses the URL content to obtain the embedded object prior to receiving a corresponding embedded object request message initiated by the web browser,
 wherein the upstream proxy server comprises means for using a cookie when obtaining the embedded object.

* * * * *



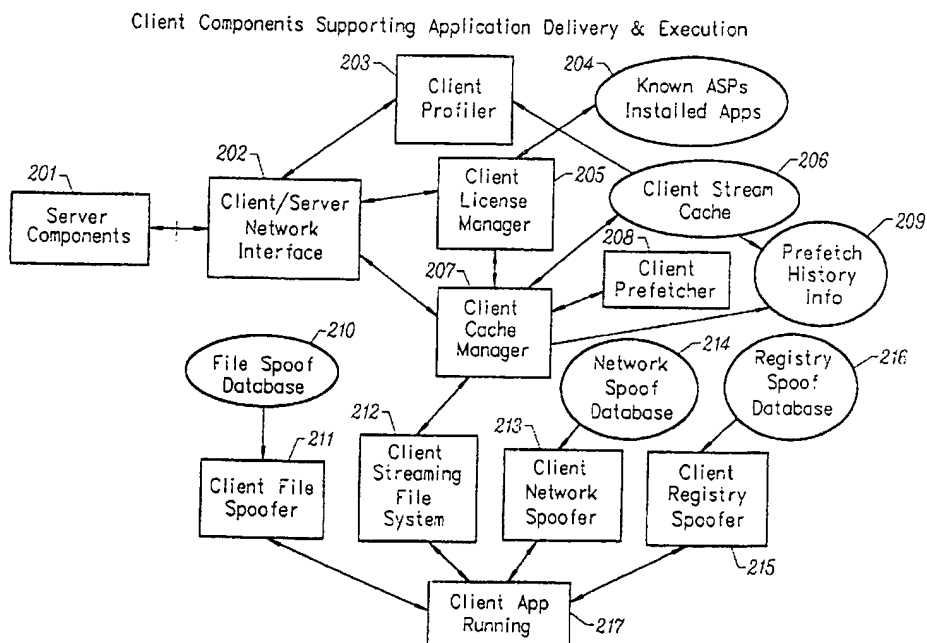
US 20020091763A1

(19) **United States**(12) **Patent Application Publication**
Shah et al.(10) Pub. No.: **US 2002/0091763 A1**
(43) Pub. Date: **Jul. 11, 2002**(54) **CLIENT-SIDE PERFORMANCE
OPTIMIZATION SYSTEM FOR STREAMED
APPLICATIONS****Publication Classification**(51) Int. Cl.⁷ G06F 15/16

(52) U.S. Cl. 709/203

(76) Inventors: **Lucky Vasant Shah**, Fremont, CA
(US); **Daniel Takeo Arai**, Sunnyvale,
CA (US); **Manuel Enrique Benitez**,
Cupertino, CA (US); **Anne Marie
Holler**, Santa Clara, CA (US); **Robert
Curtis Wohlgenuth**, Santa Clara, CA
(US)Correspondence Address:
GLENN PATENT GROUP
3475 EDISON WAY
SUITE L
MENLO PARK, CA 94025 (US)(21) Appl. No.: **09/858,260**(22) Filed: **May 15, 2001****Related U.S. Application Data**(63) Non-provisional of provisional application No.
60/246,384, filed on Nov. 6, 2000.(57) **ABSTRACT**

An client-side performance optimization system for streamed applications provides several approaches for fulfilling client-side application code and data file requests for streamed applications. A streaming file system or file driver is installed on the client system that receives and fulfills application code and data requests from a persistent cache or the streaming application server. The client or the server can initiate the prefetching of application code and data to improve interactive application performance. A client-to-client communication mechanism allows local application customization to travel from one client machine to another without involving server communication. Applications are patched or upgraded via a change in the root directory for that application. The client can be notified of application upgrades by the server which can be marked as mandatory, in which case the client will force the application to be upgraded. The server broadcasts an application program's code and data and any client that is interested in that particular application program stores the broadcasted code and data for later use.



Server Components Supporting Application Delivery & Execution License

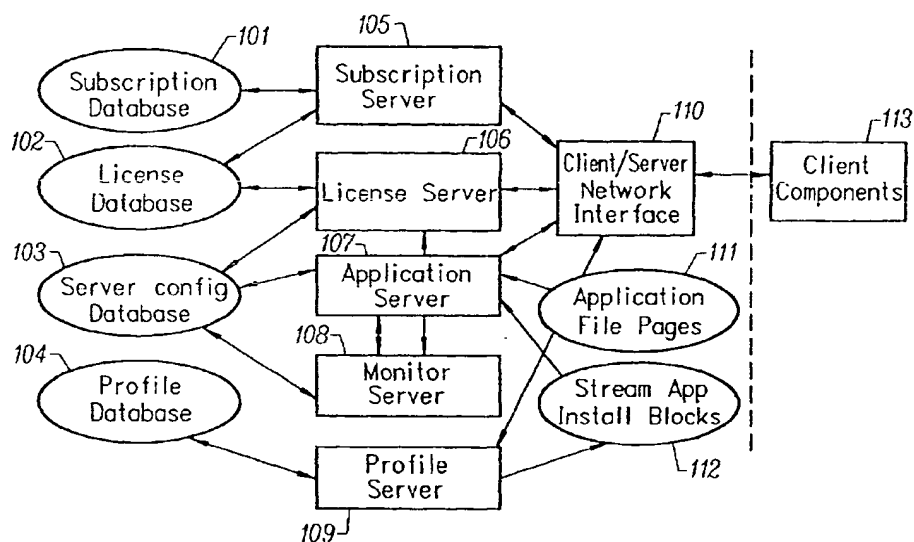


FIG. 1

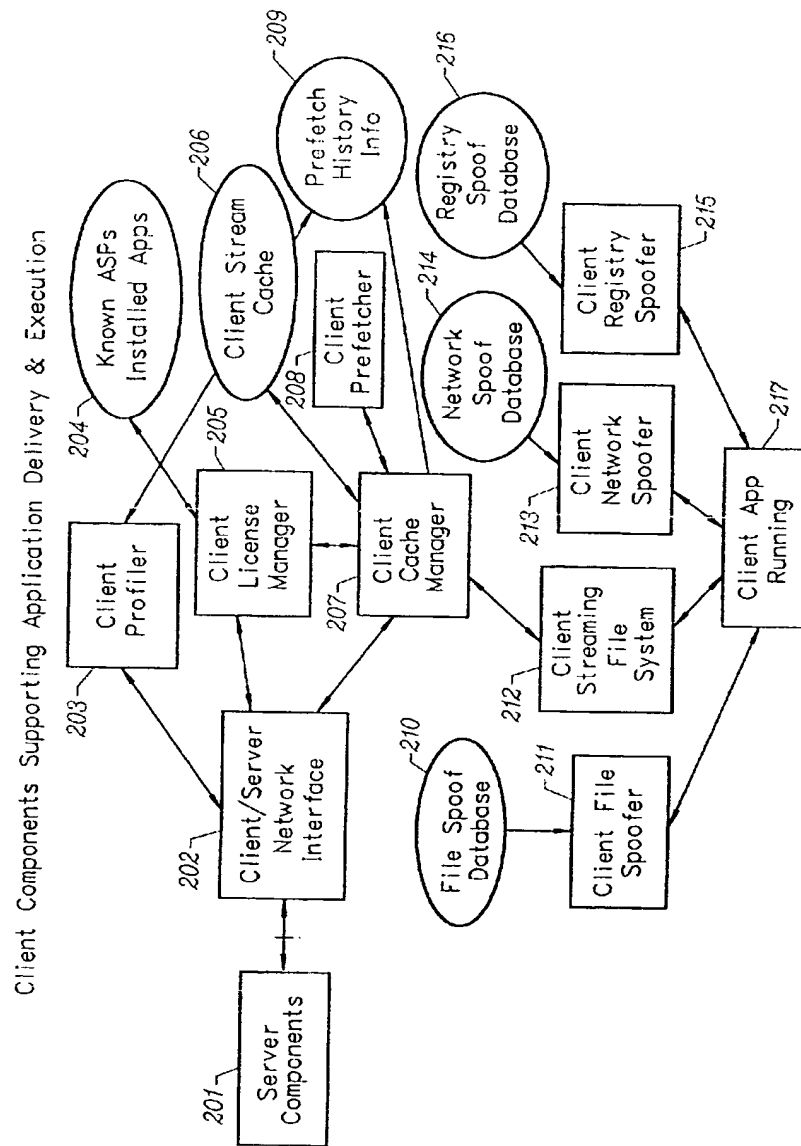


FIG. 2

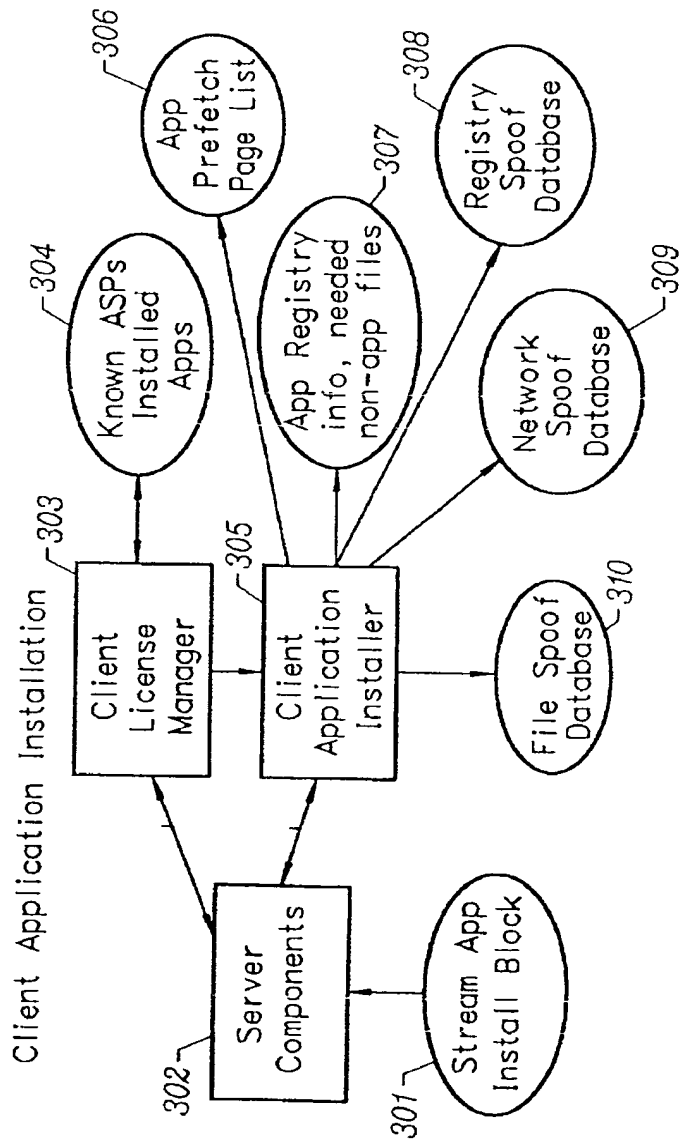


FIG. 3

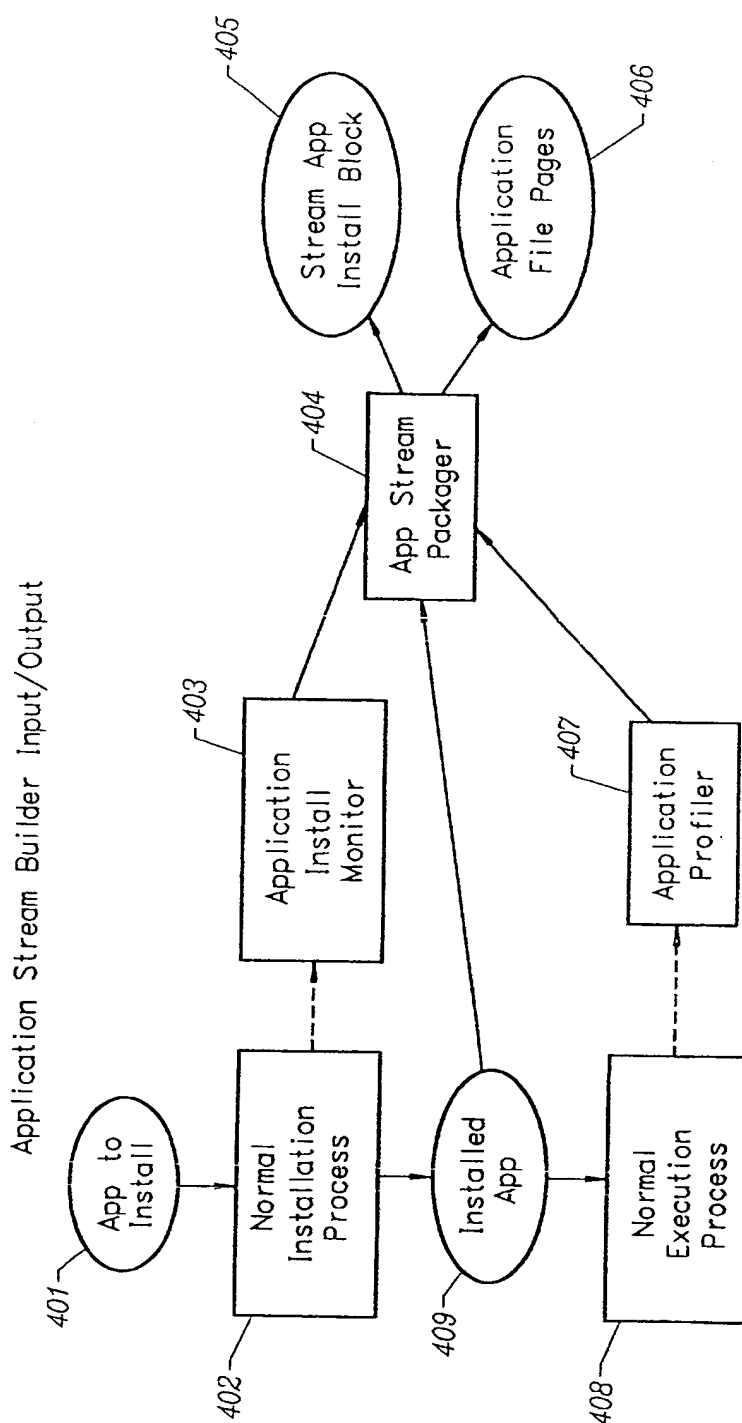


FIG. 4

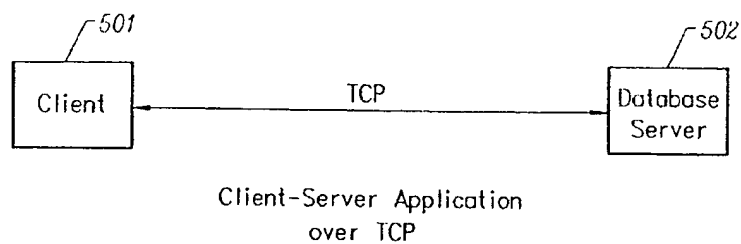


FIG. 5A

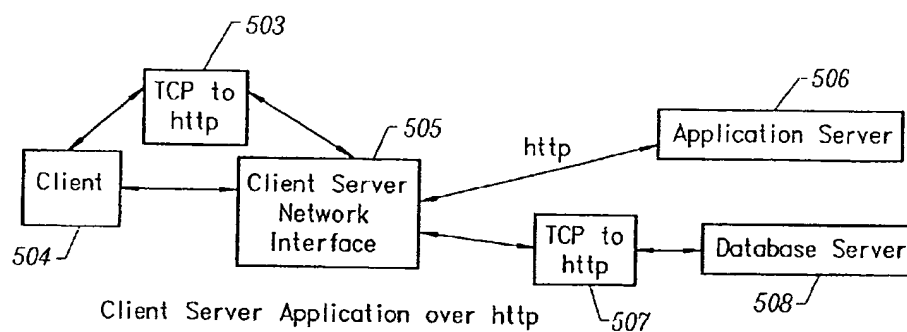


FIG. 5B

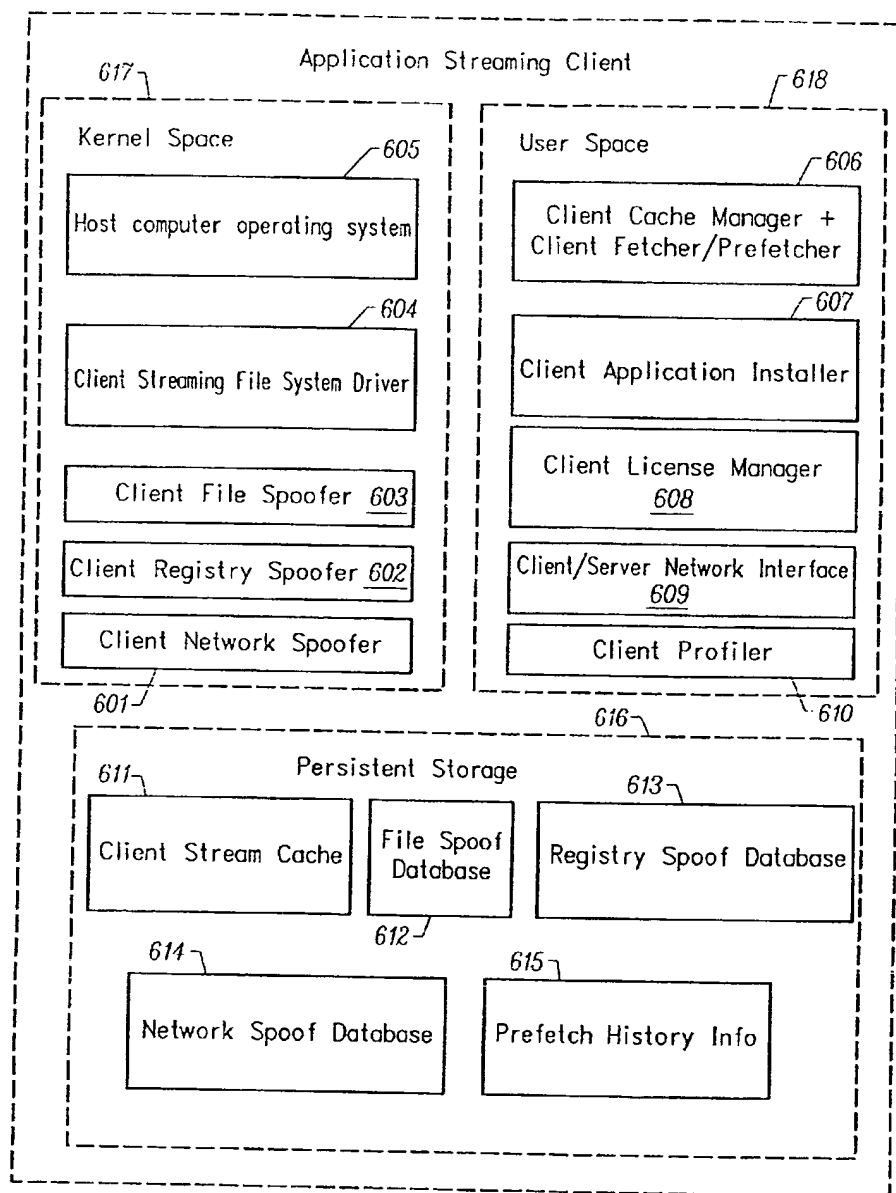


FIG. 6A

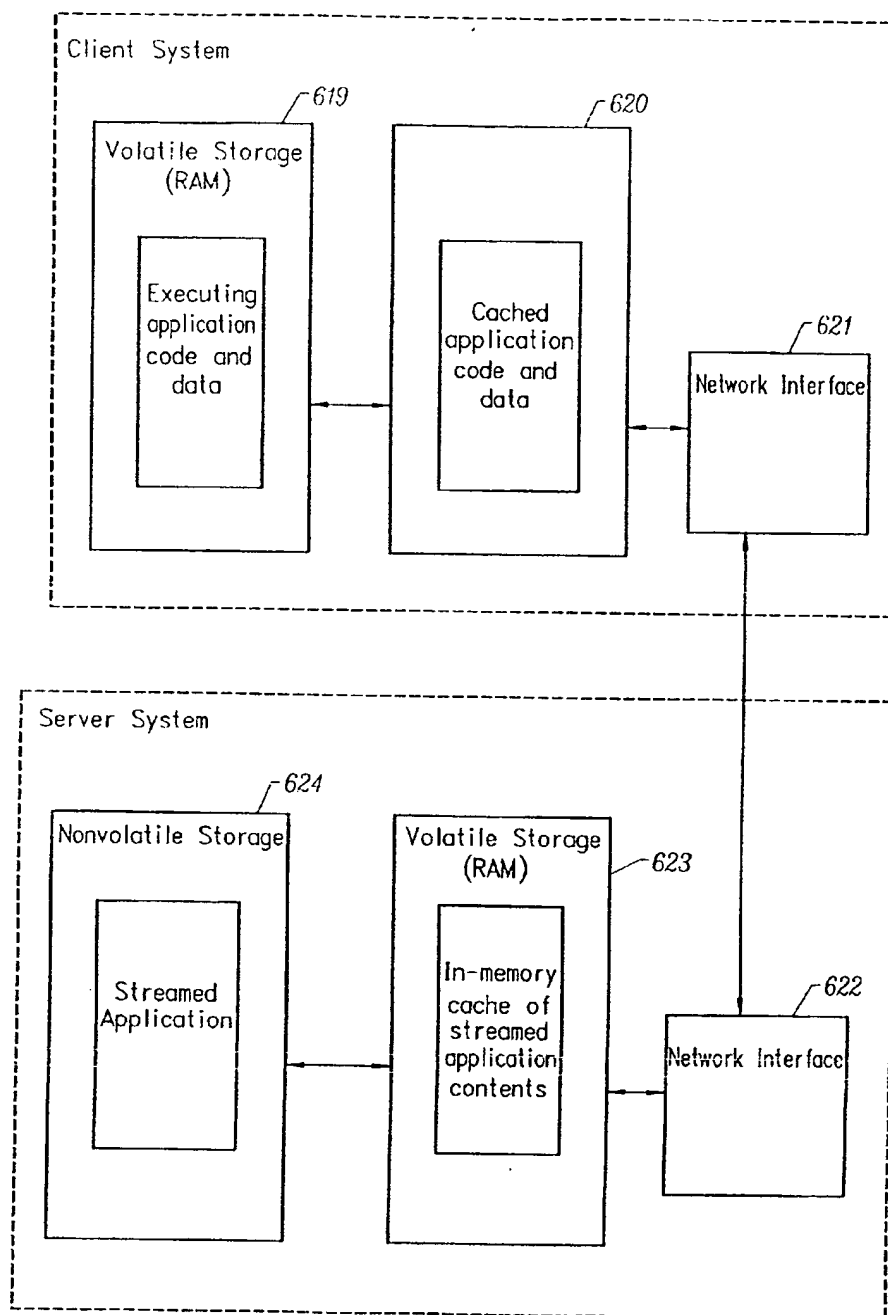


FIG. 6B

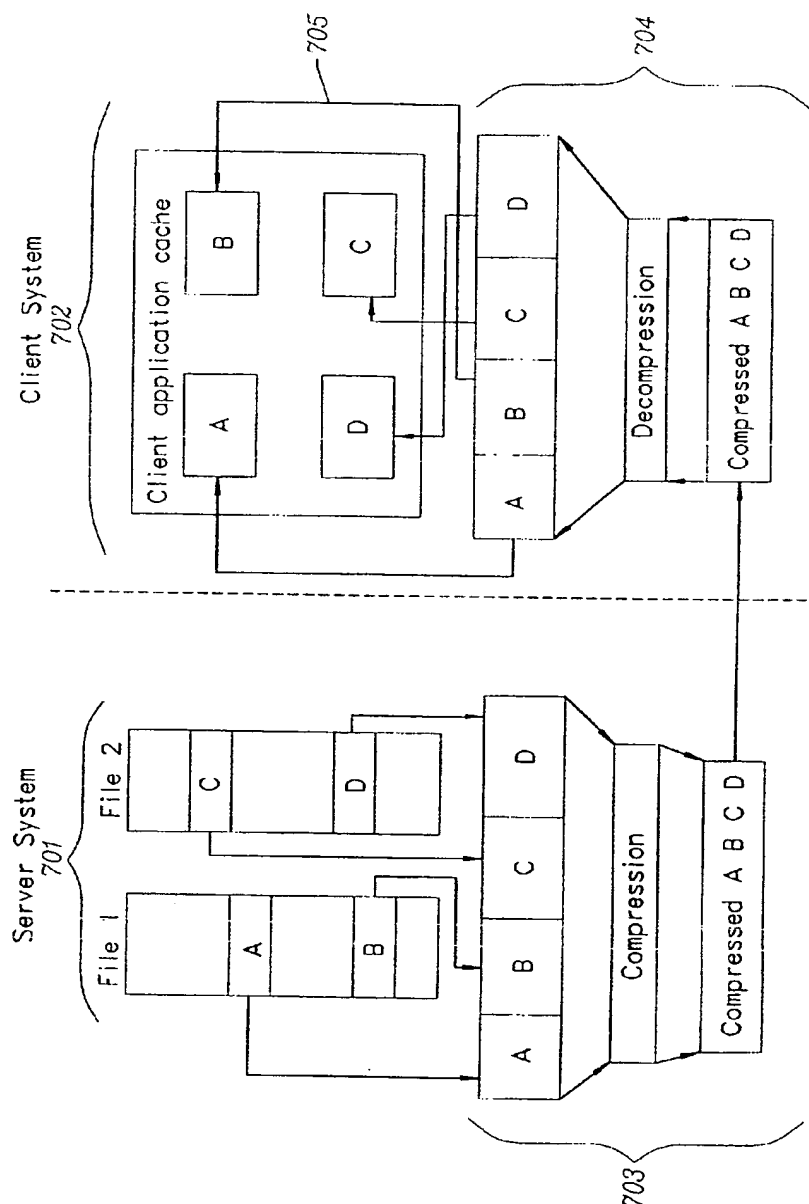


FIG. 7A

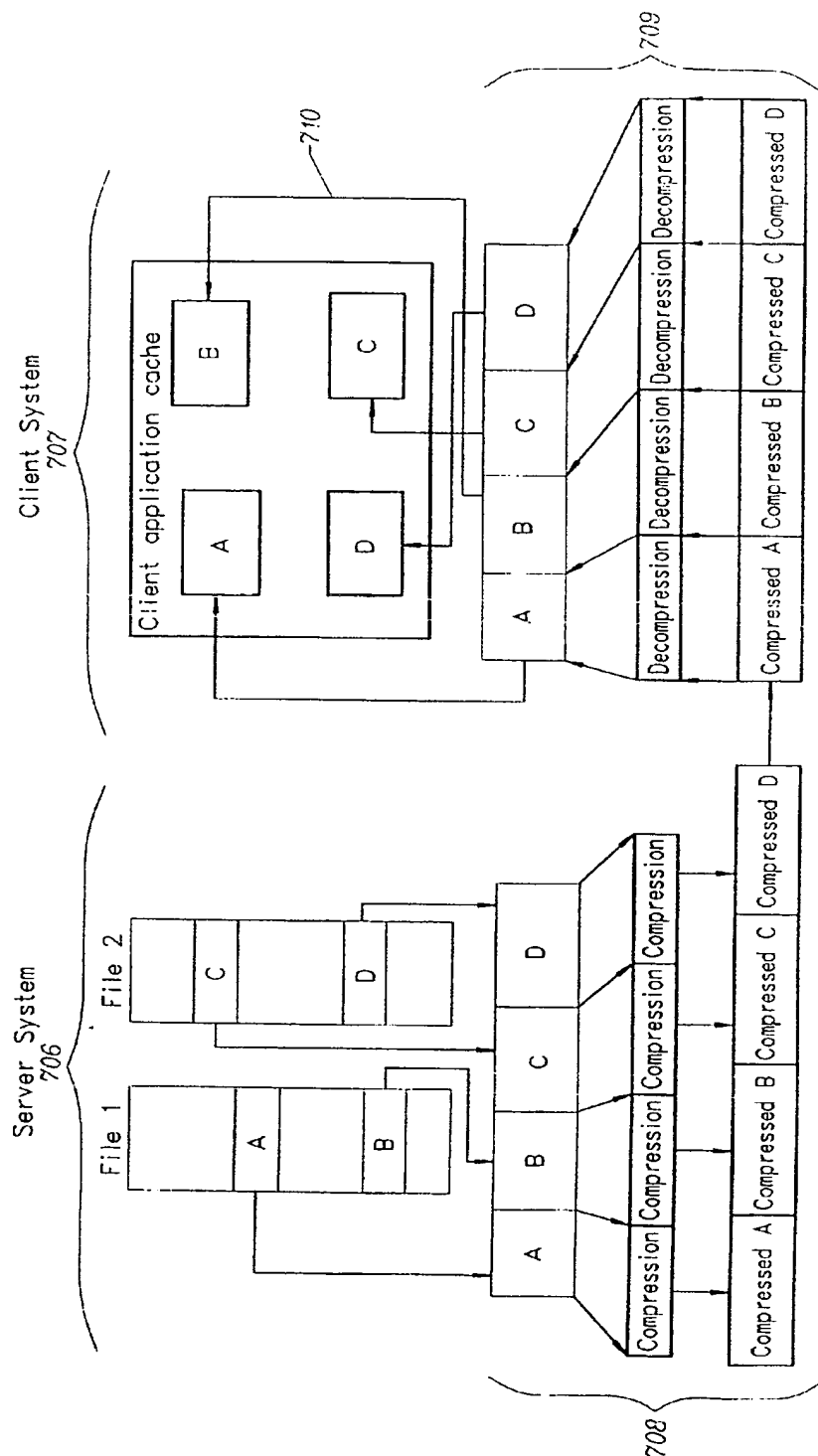


FIG. 7B

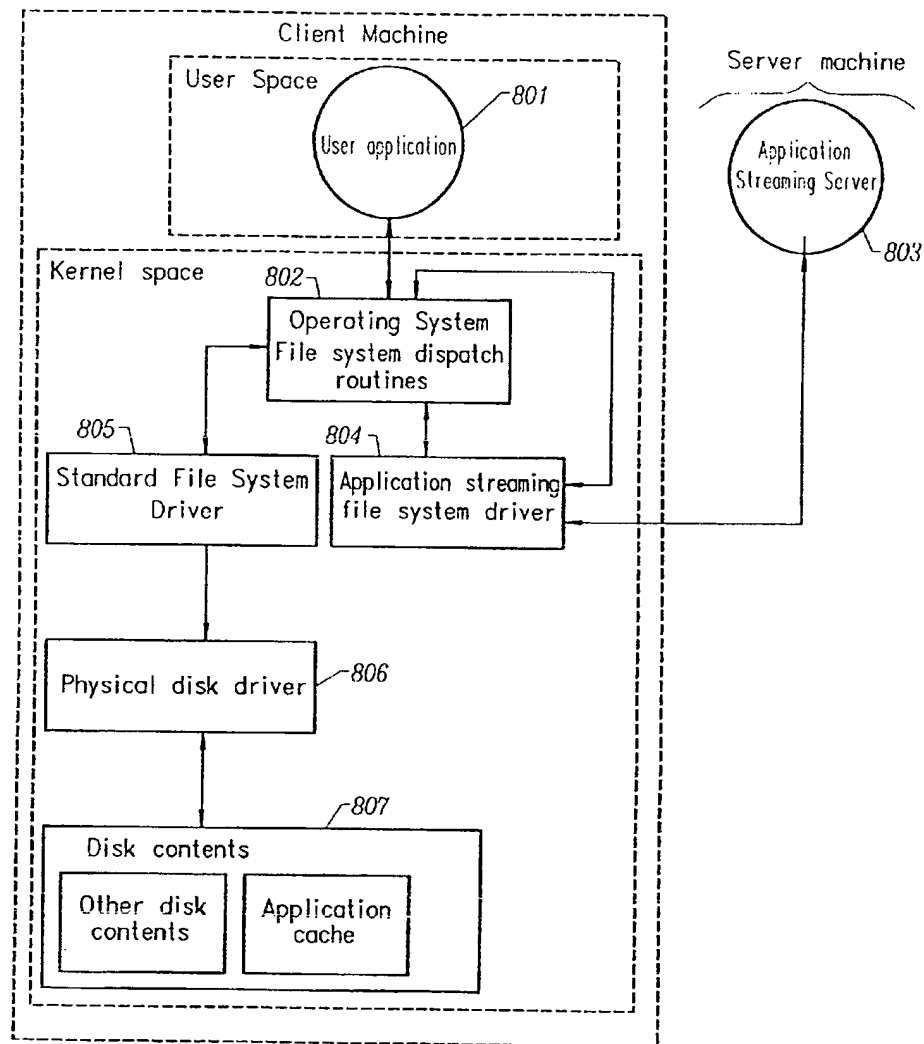


FIG. 8

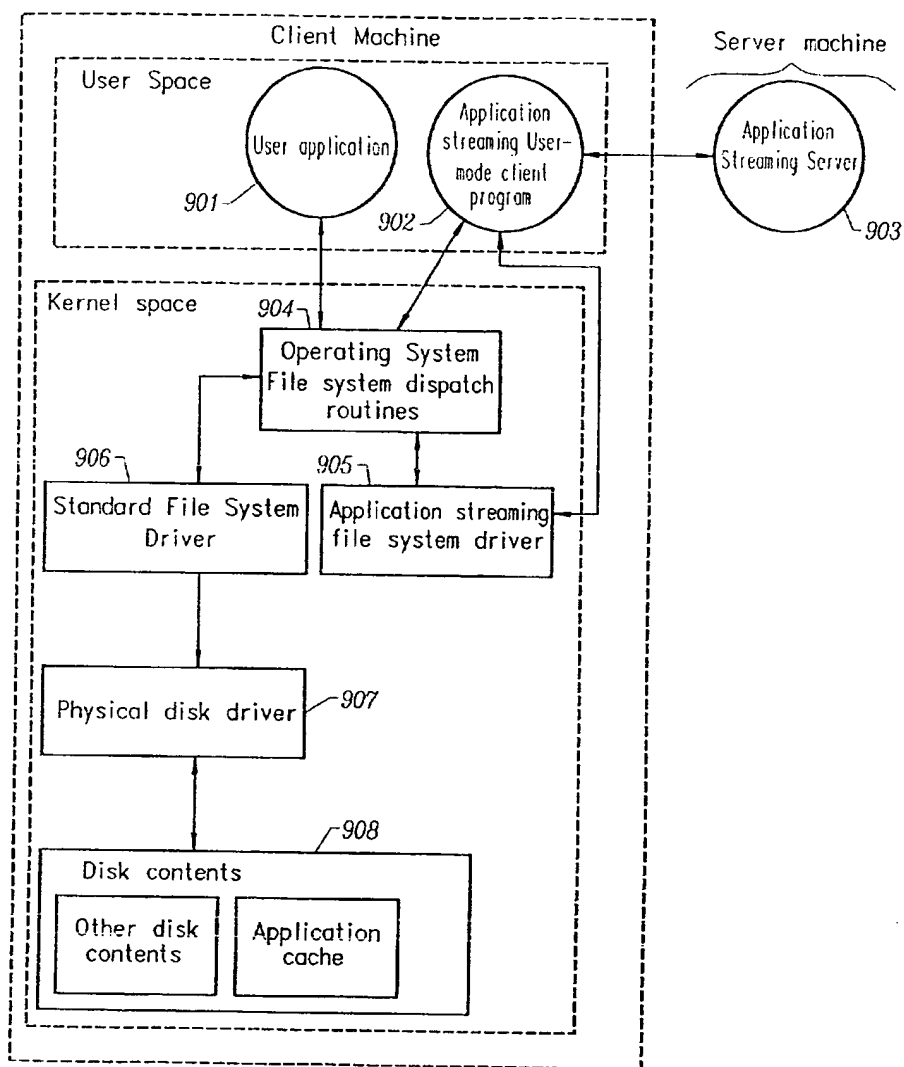


FIG. 9

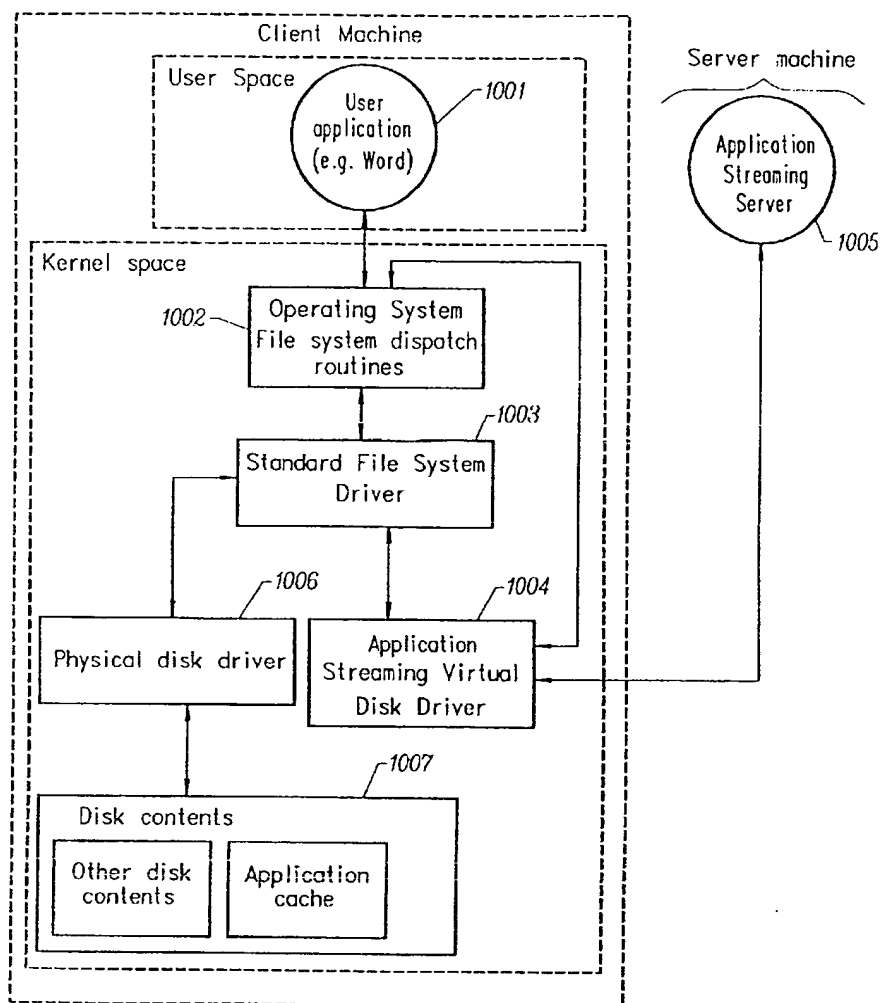


FIG. 10

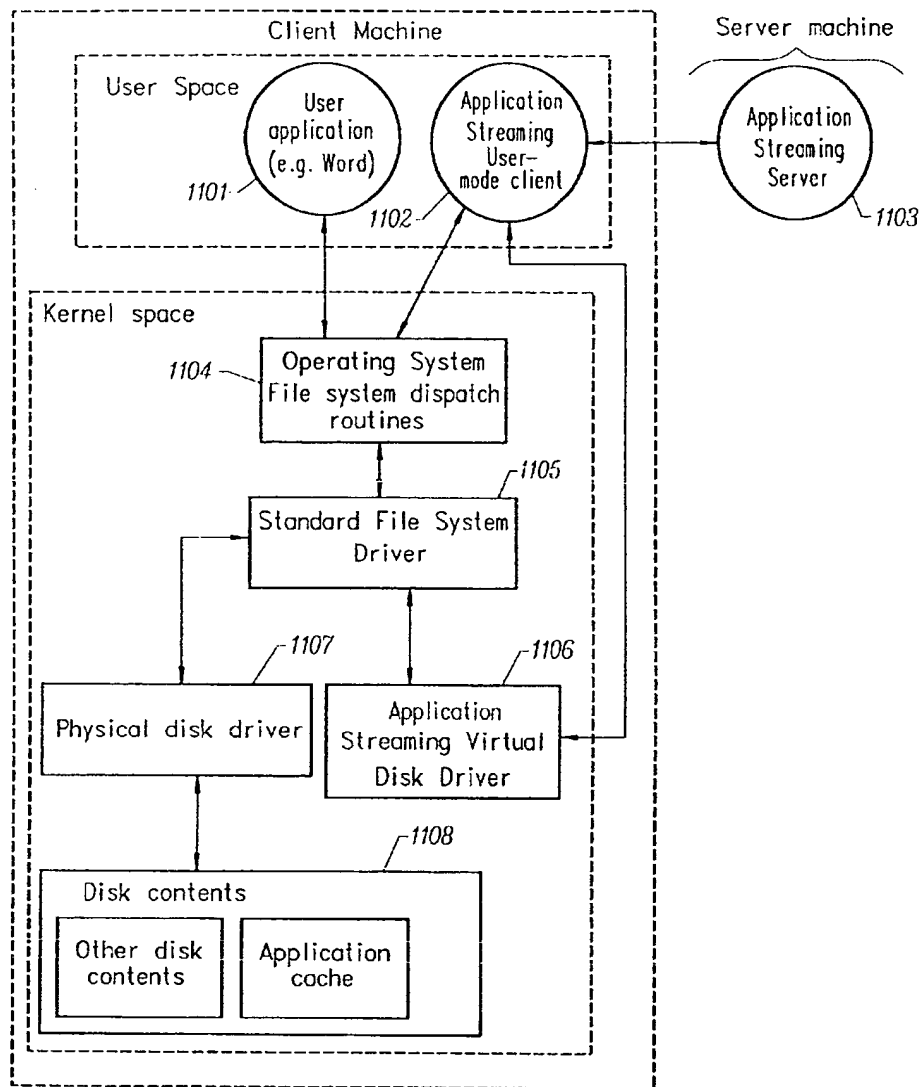


FIG. 11

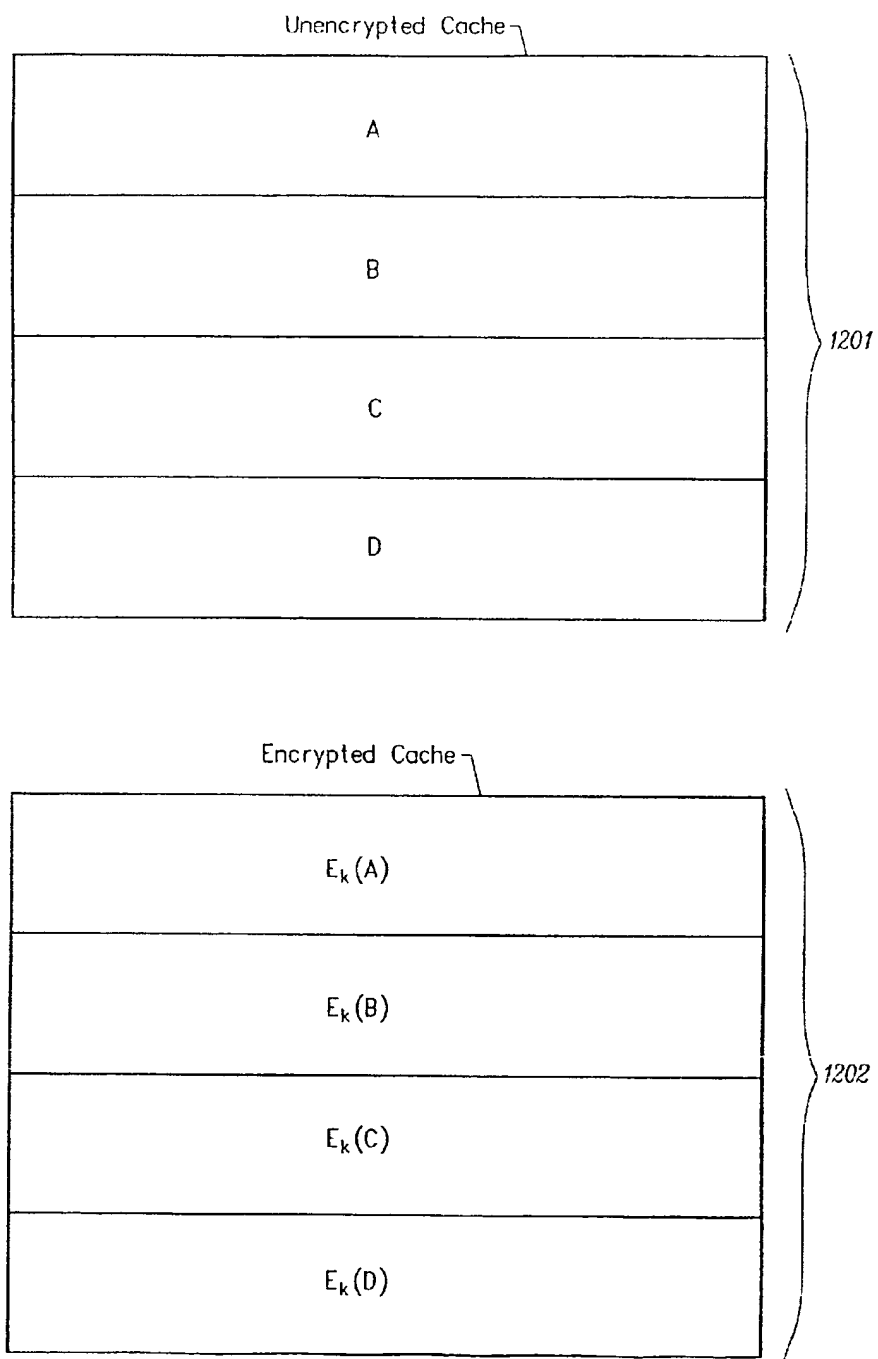


FIG. 12

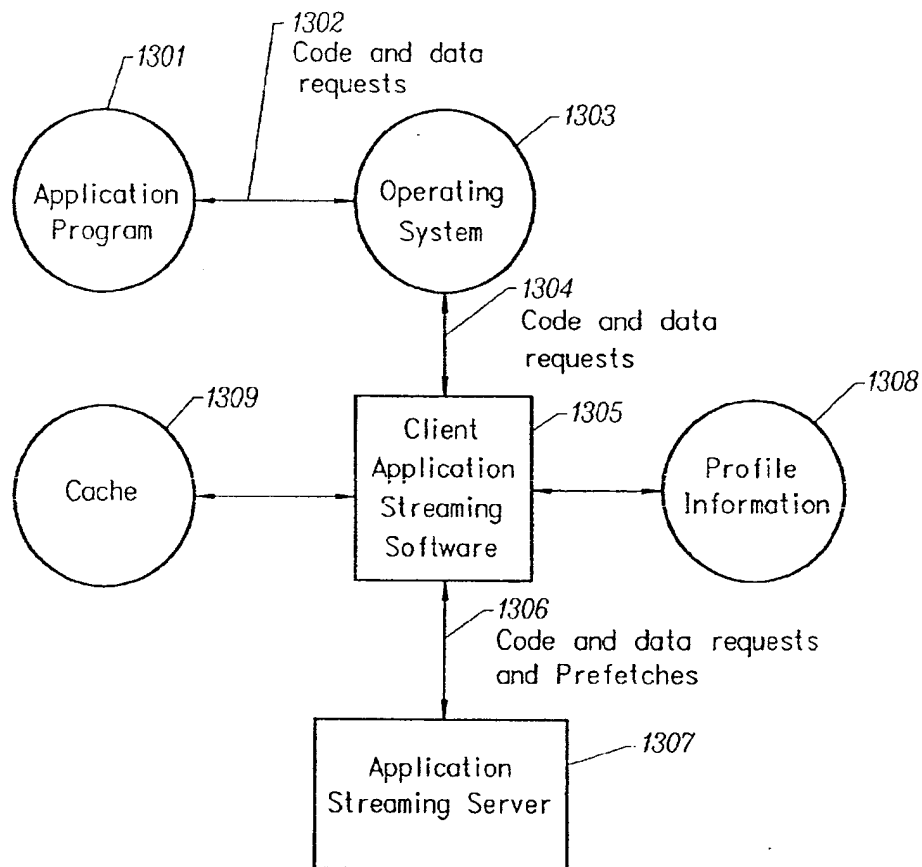


FIG. 13

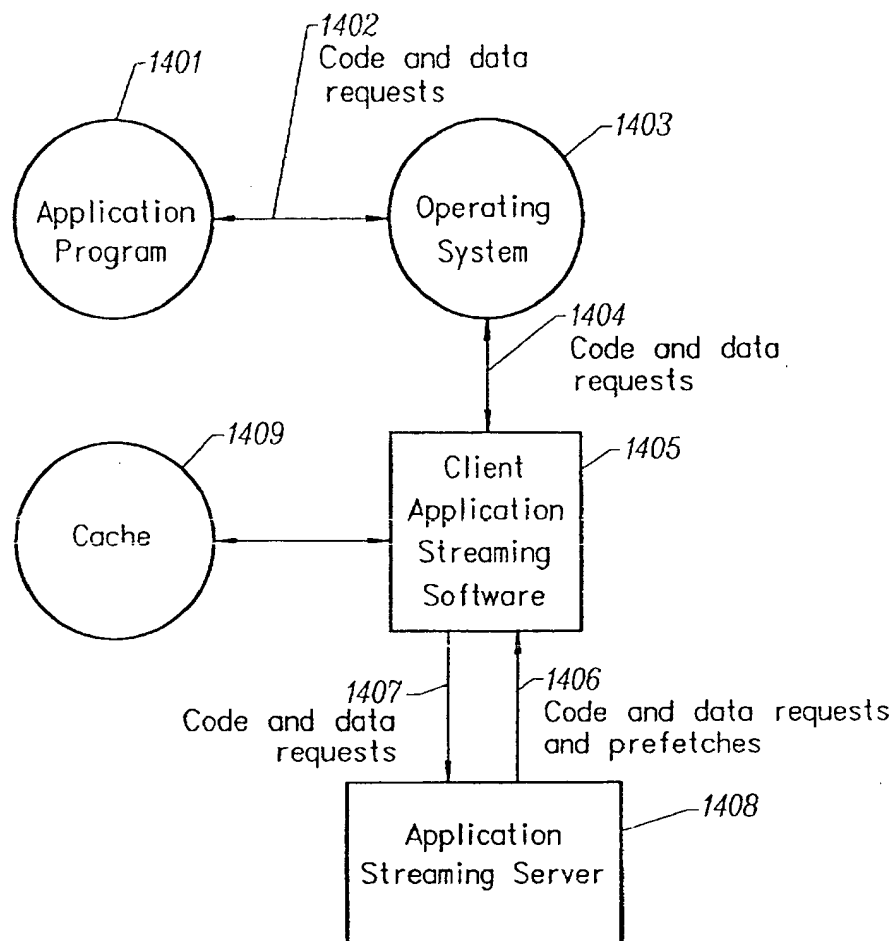
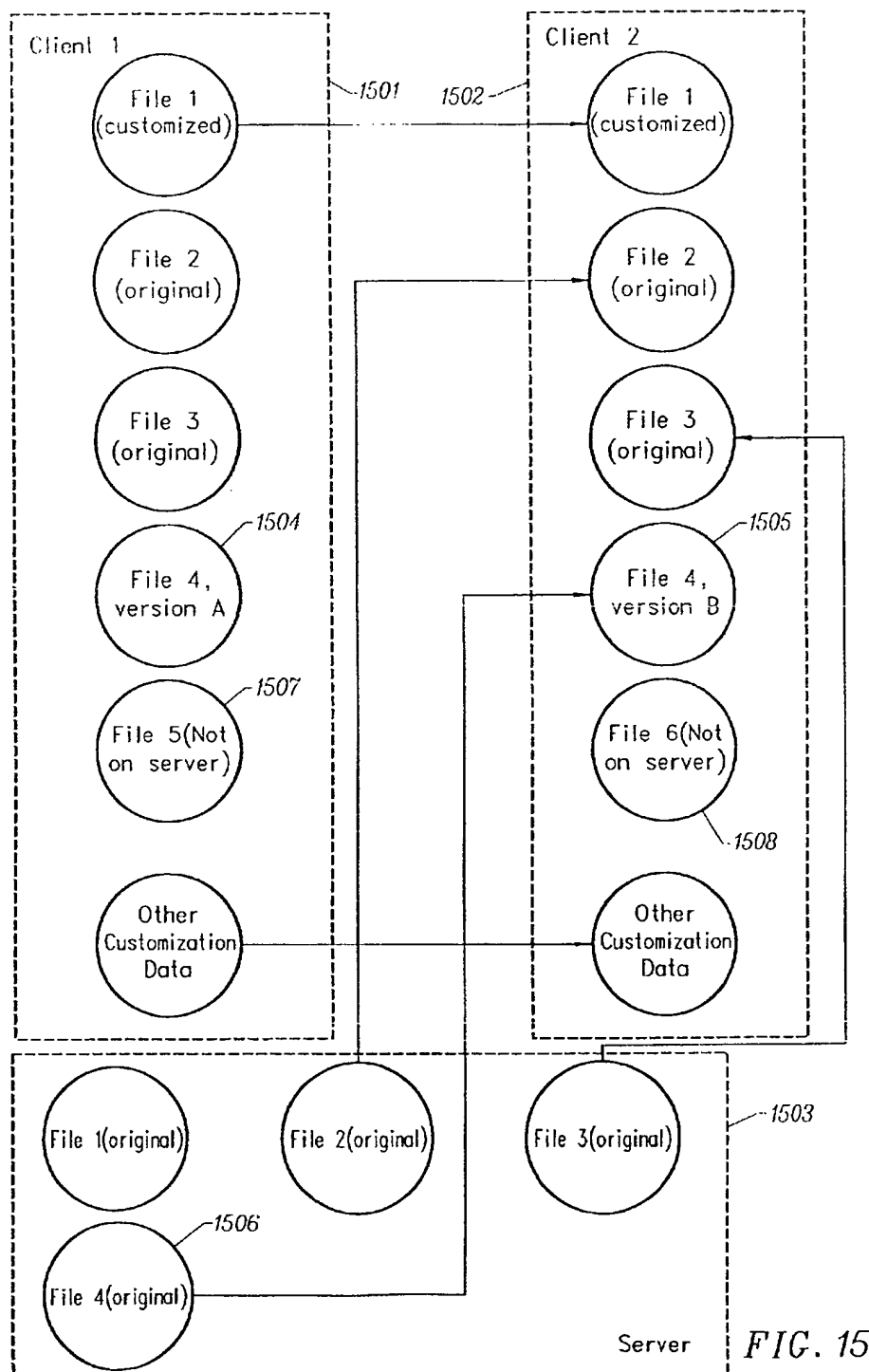


FIG. 14



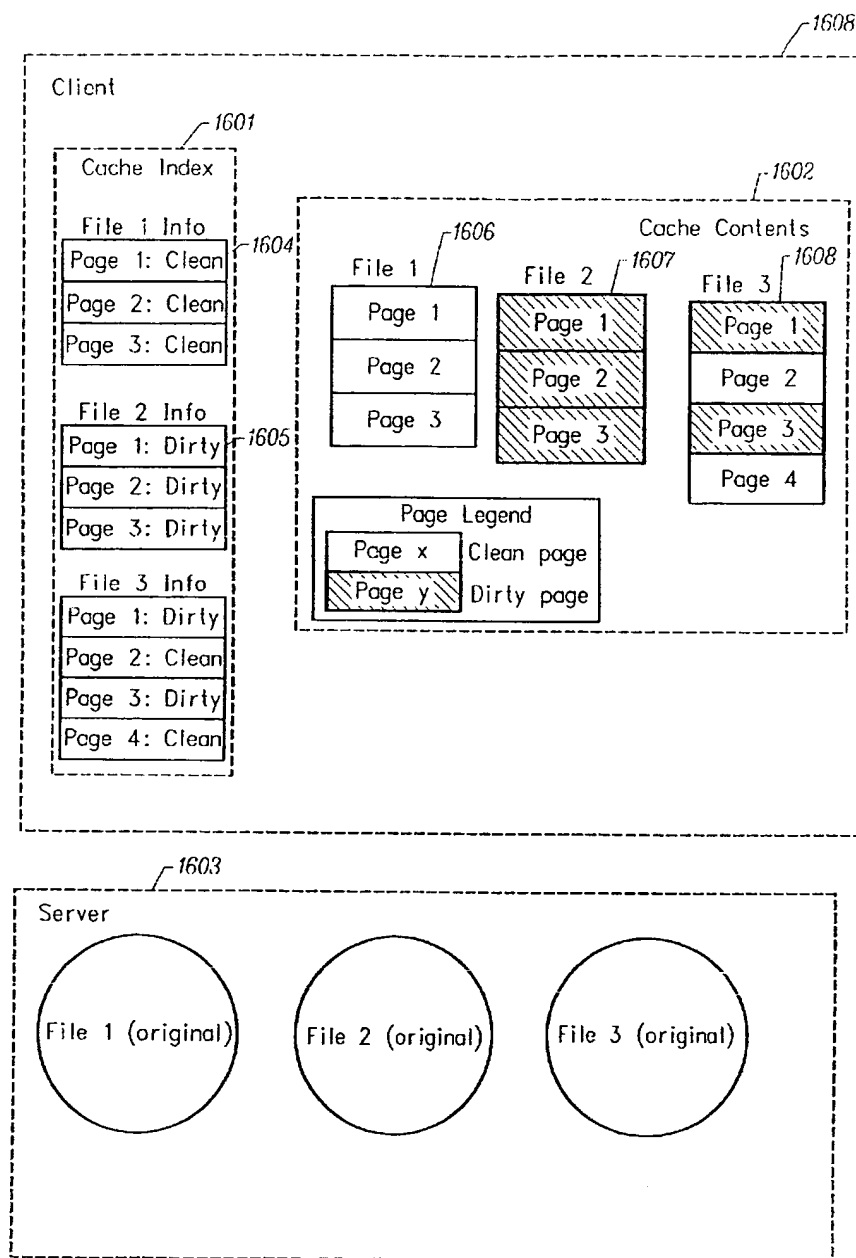


FIG. 16

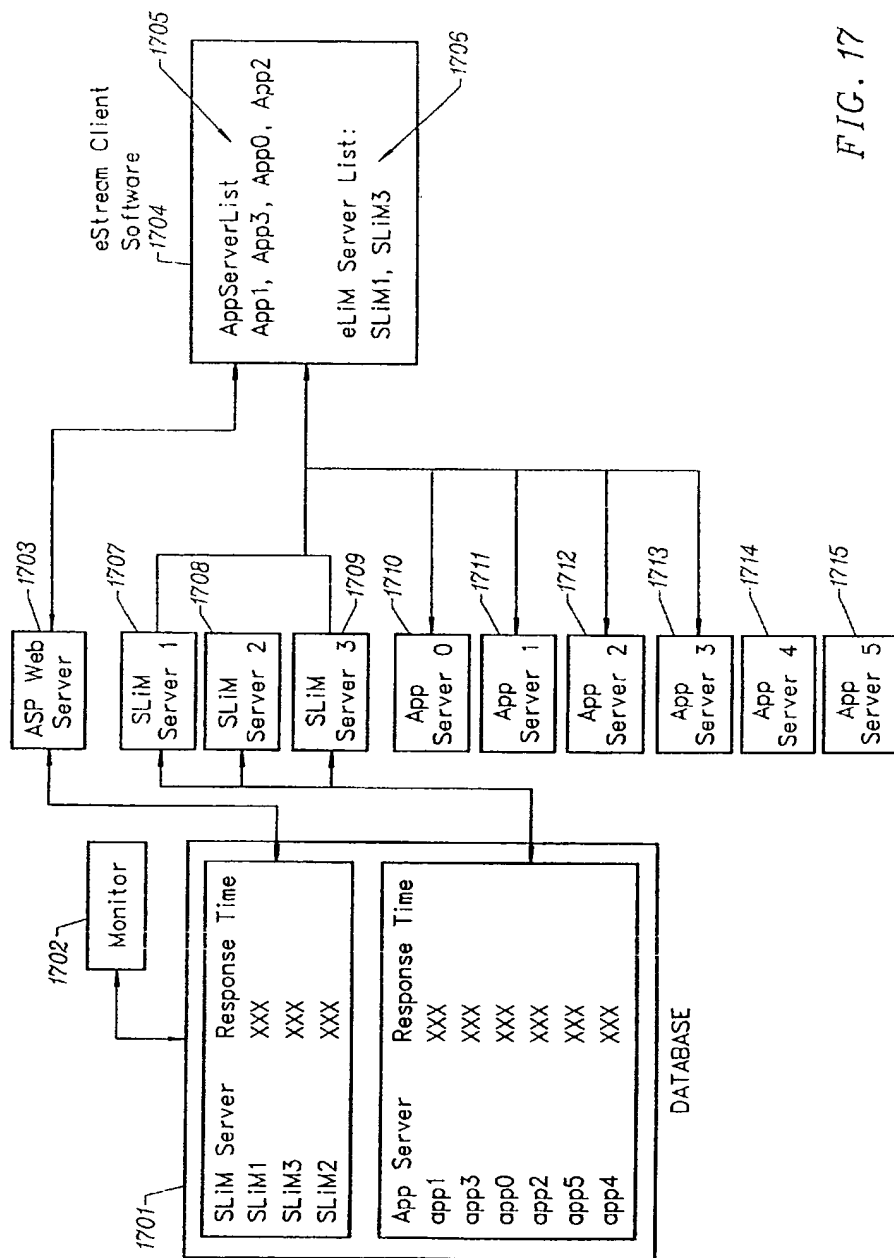
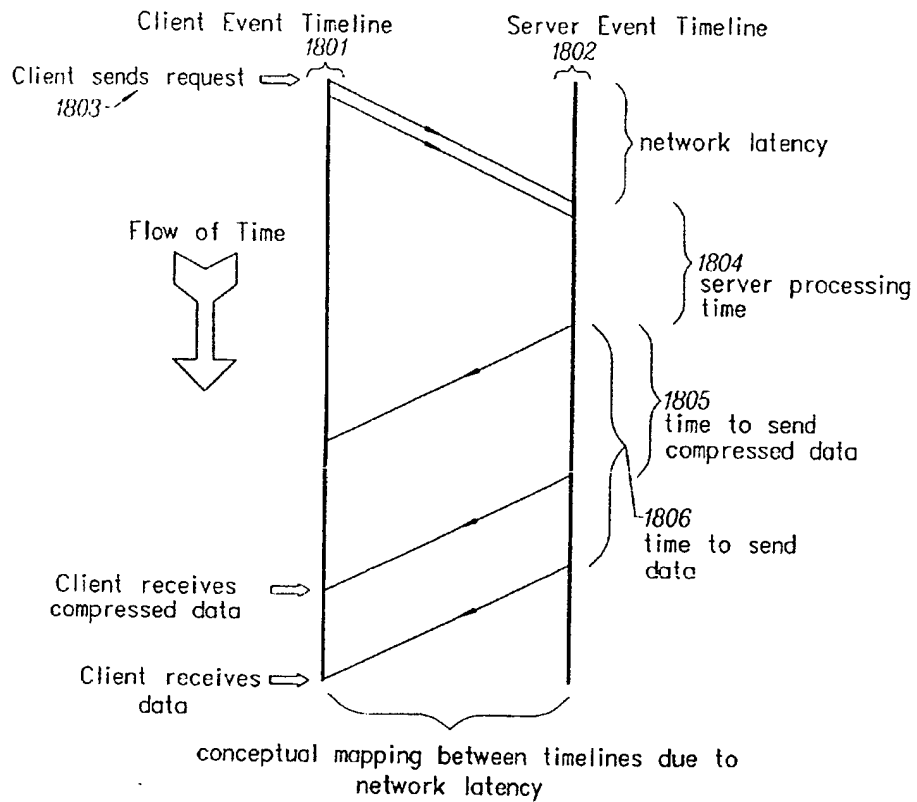


FIG. 17



Client receives data sooner if it is compressed

FIG. 18

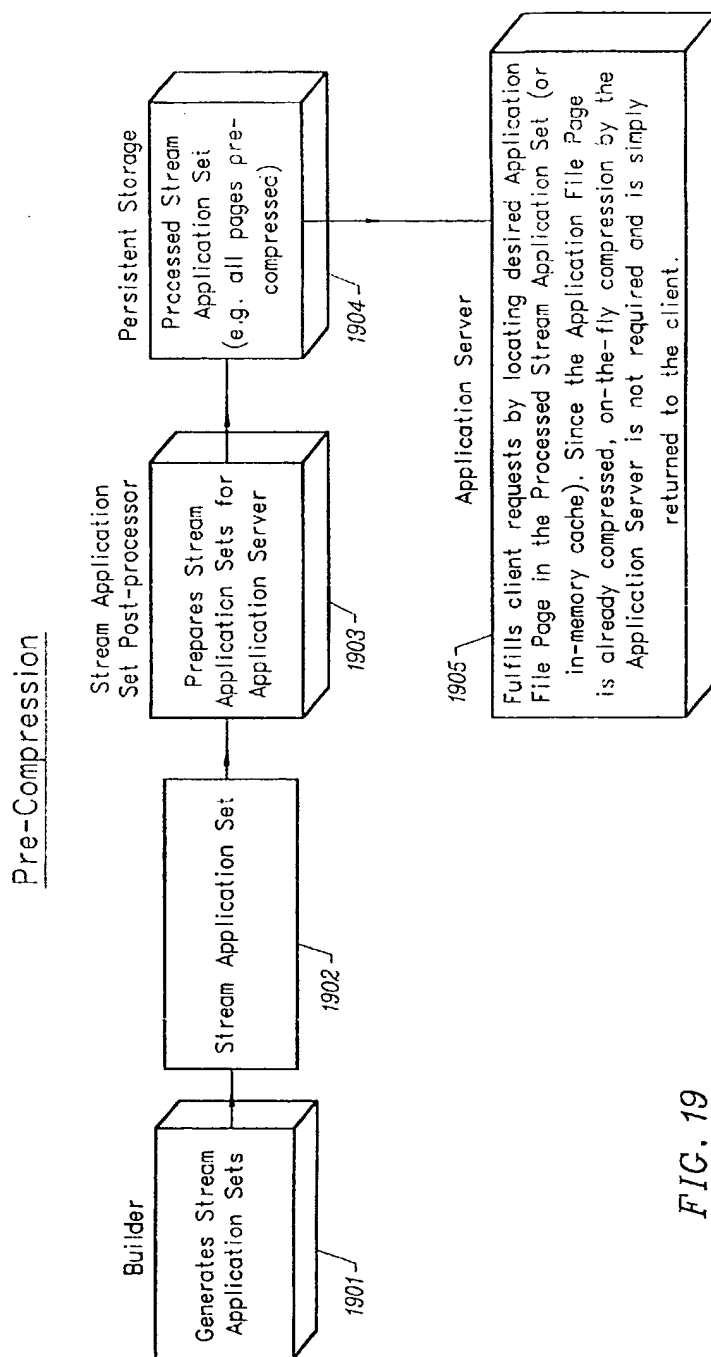
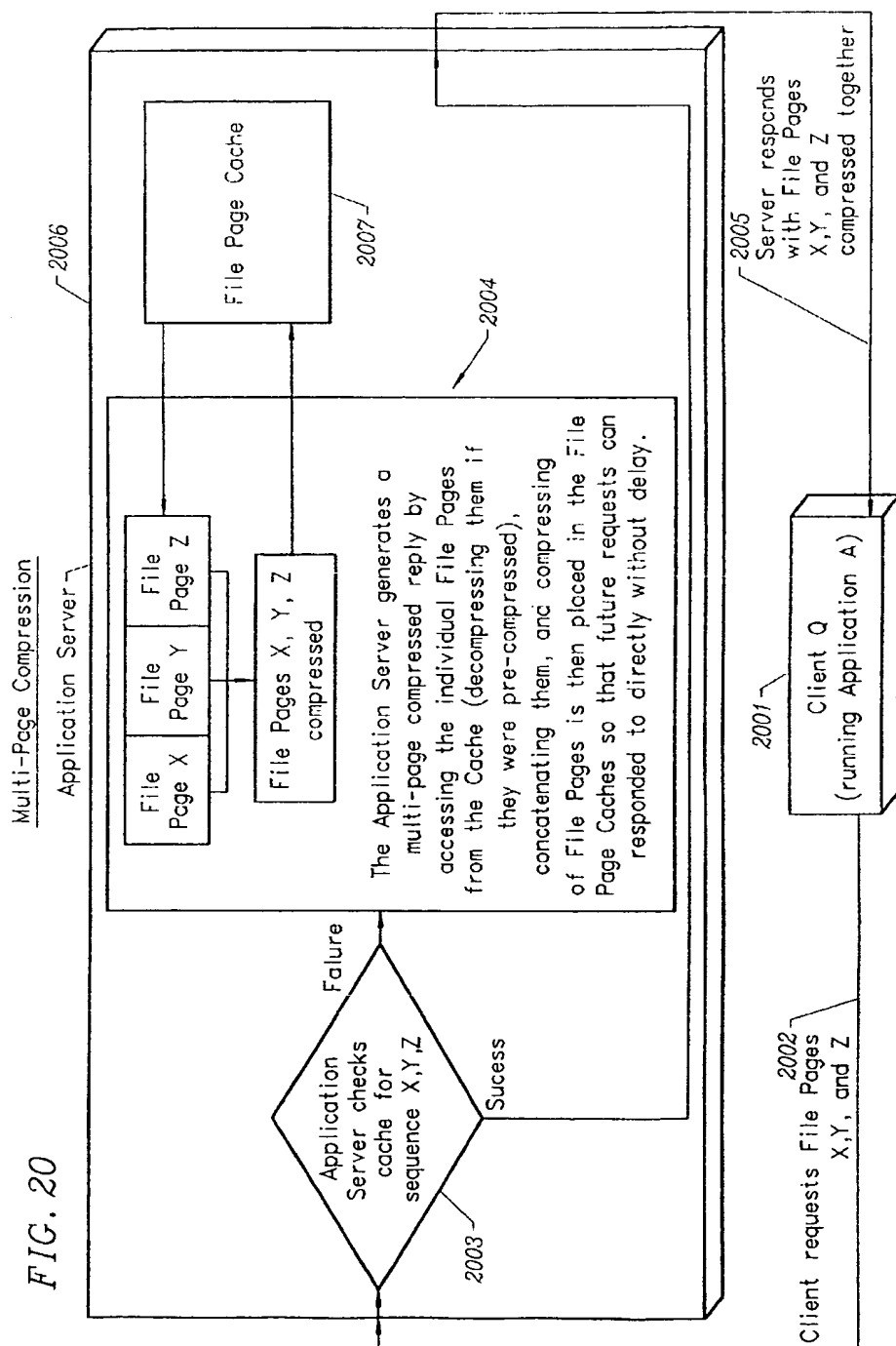
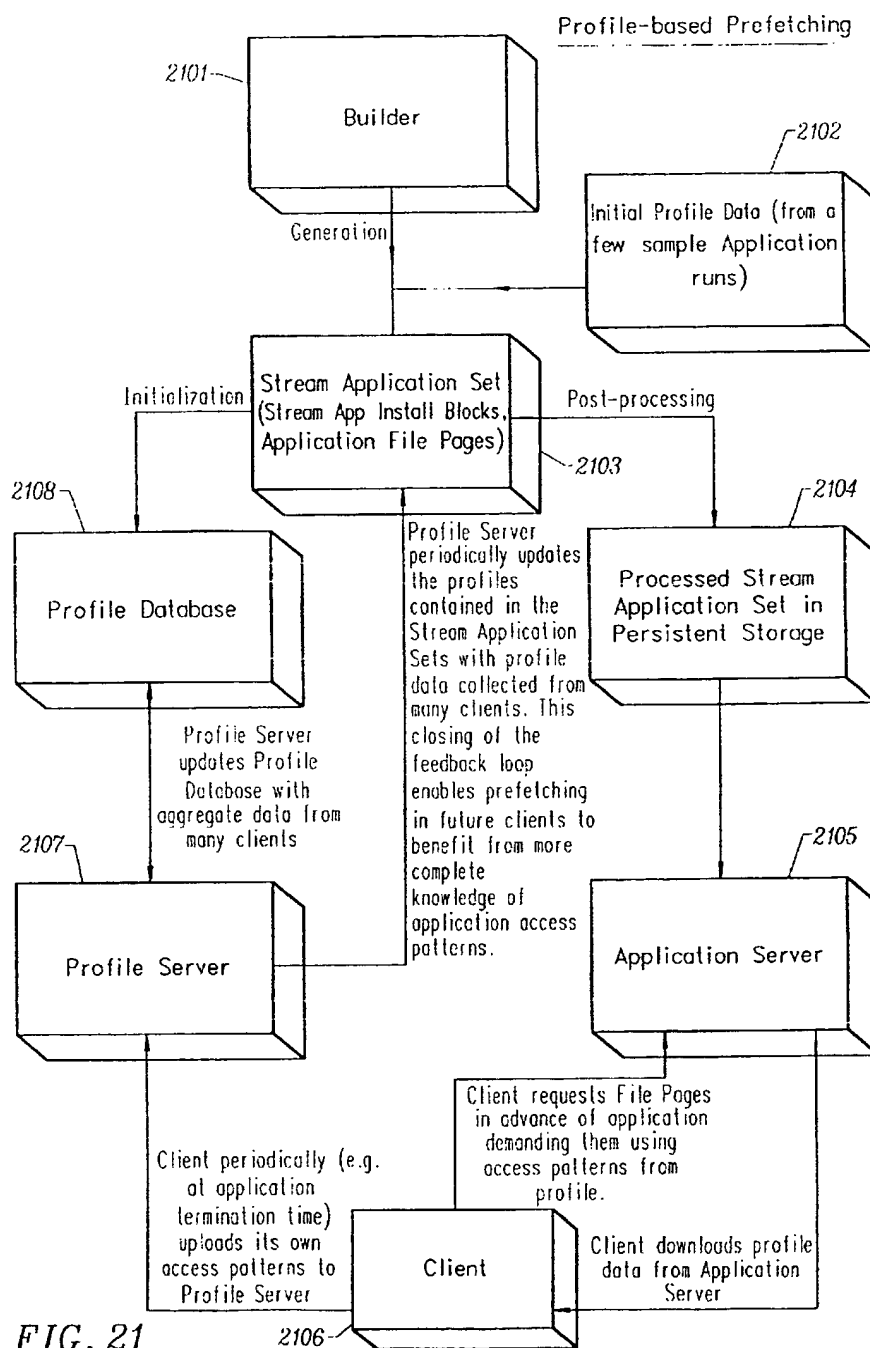


FIG. 19





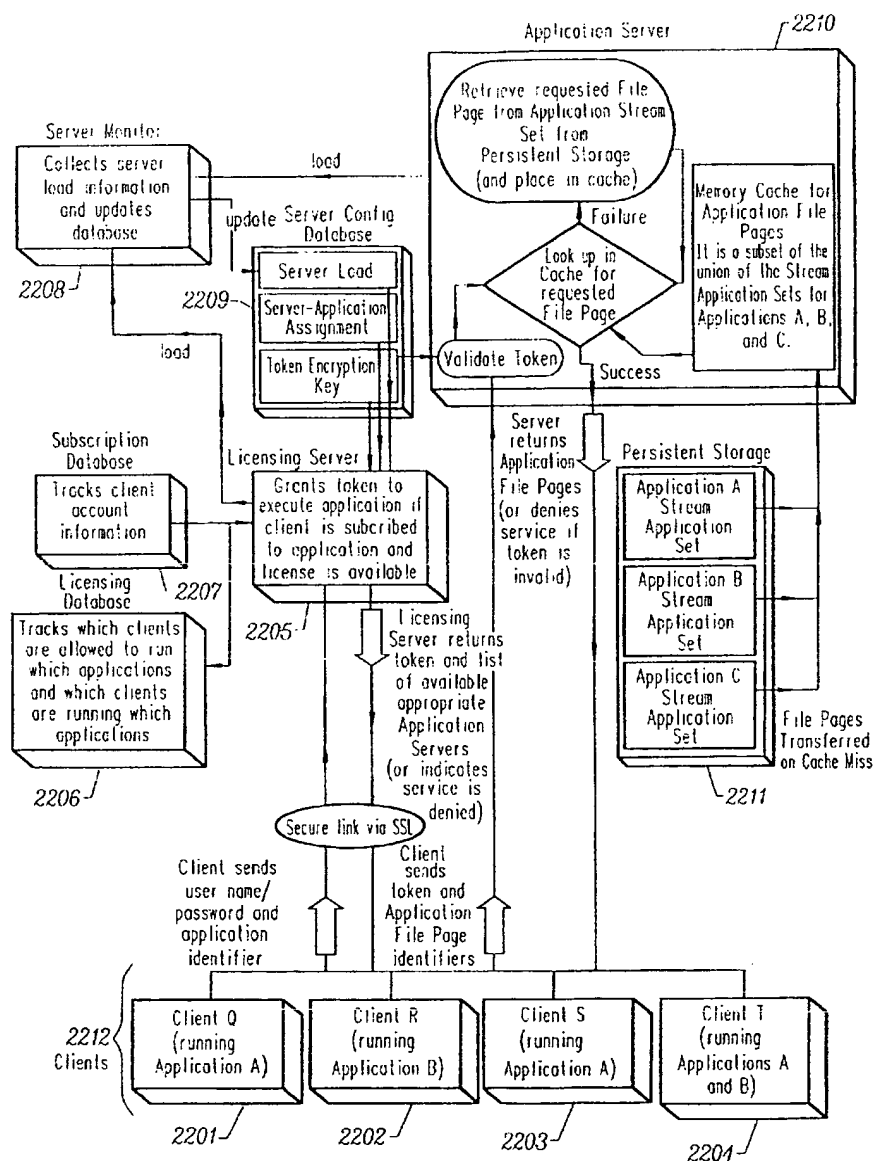


FIG. 22

Builder Install Monitor (IM) Control Flow Diagram

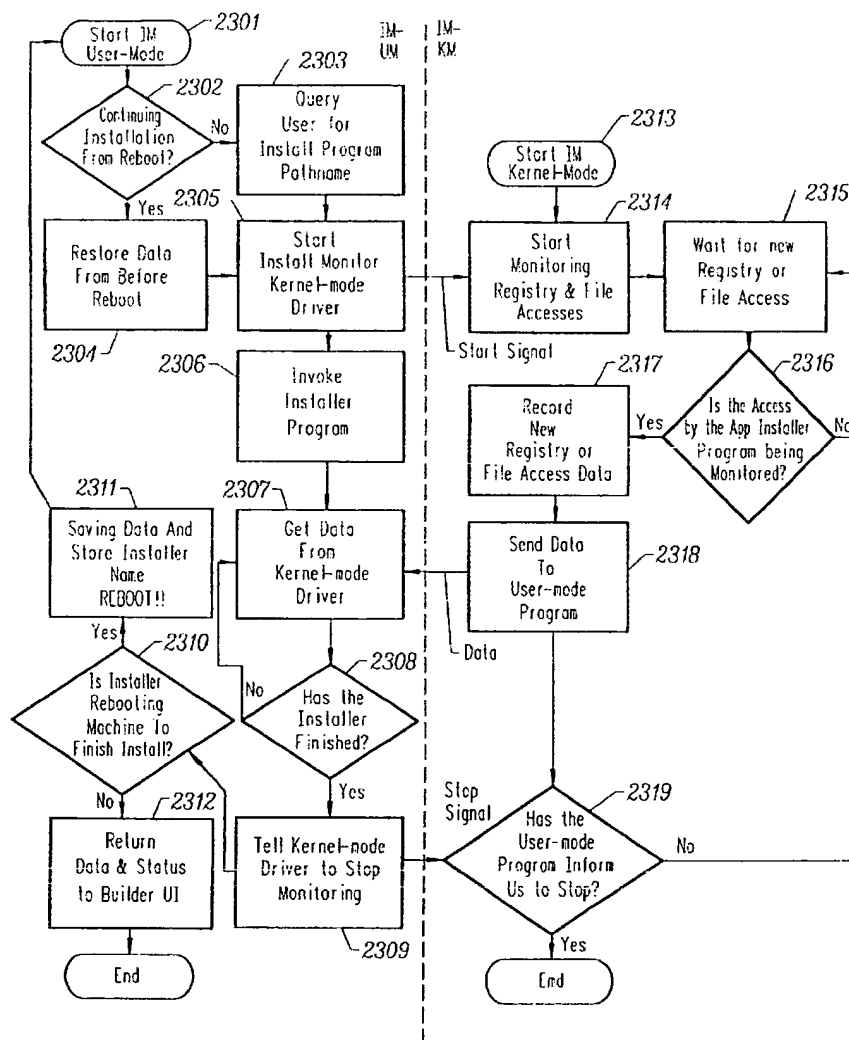


FIG. 23

Builder Application Profiler (AP) Control Flow Diagram

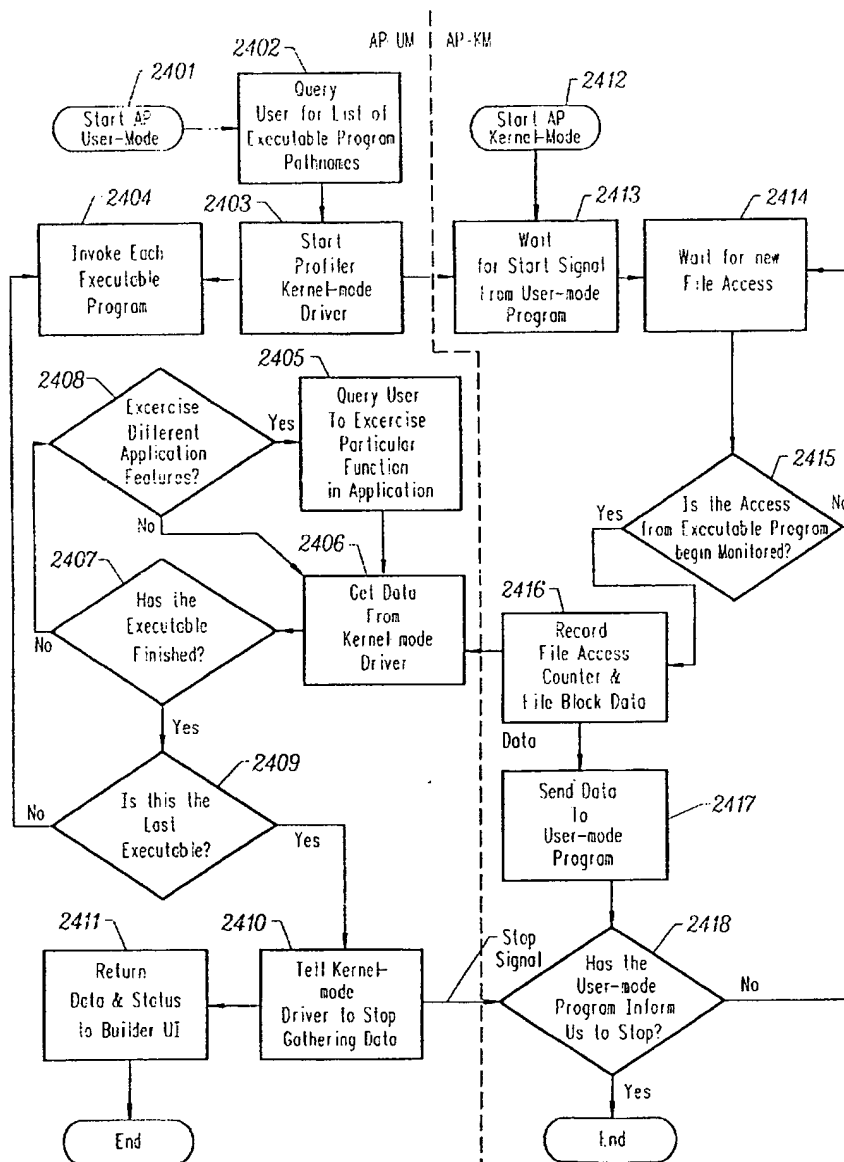


FIG. 24

Builder SAS Packager (SP) Control Flow Diagram

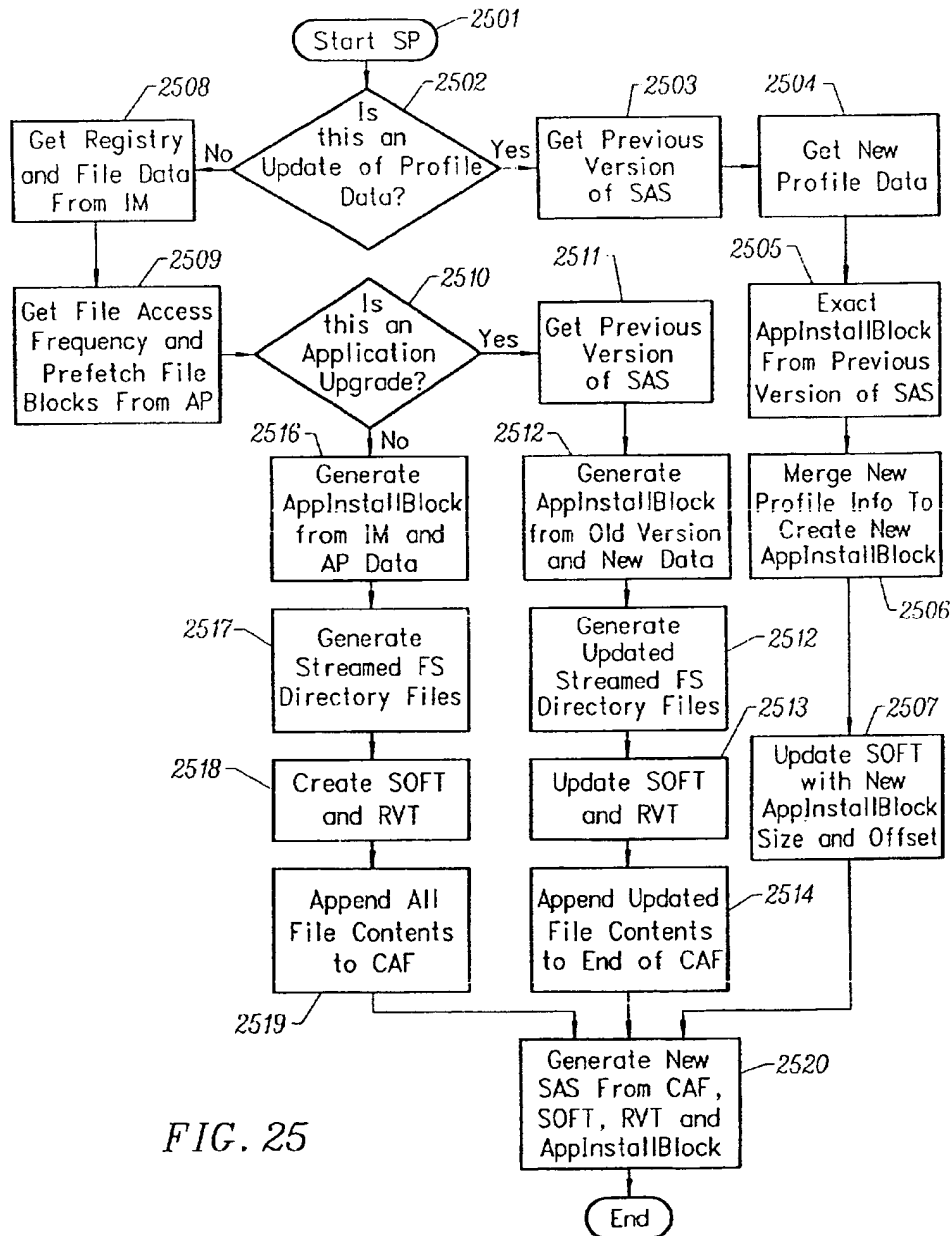


FIG. 25

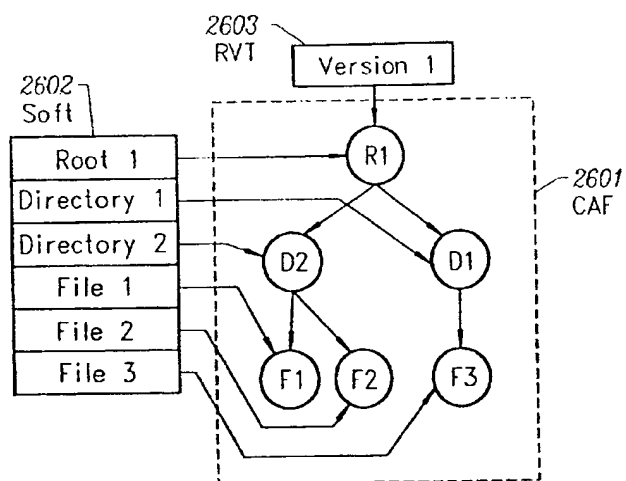


FIG. 26A

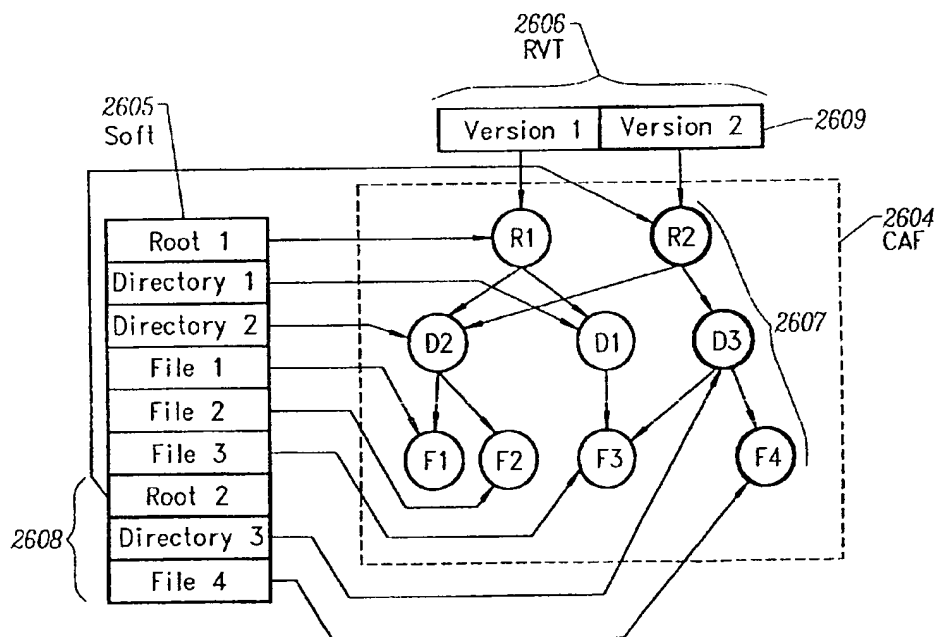


FIG. 26B

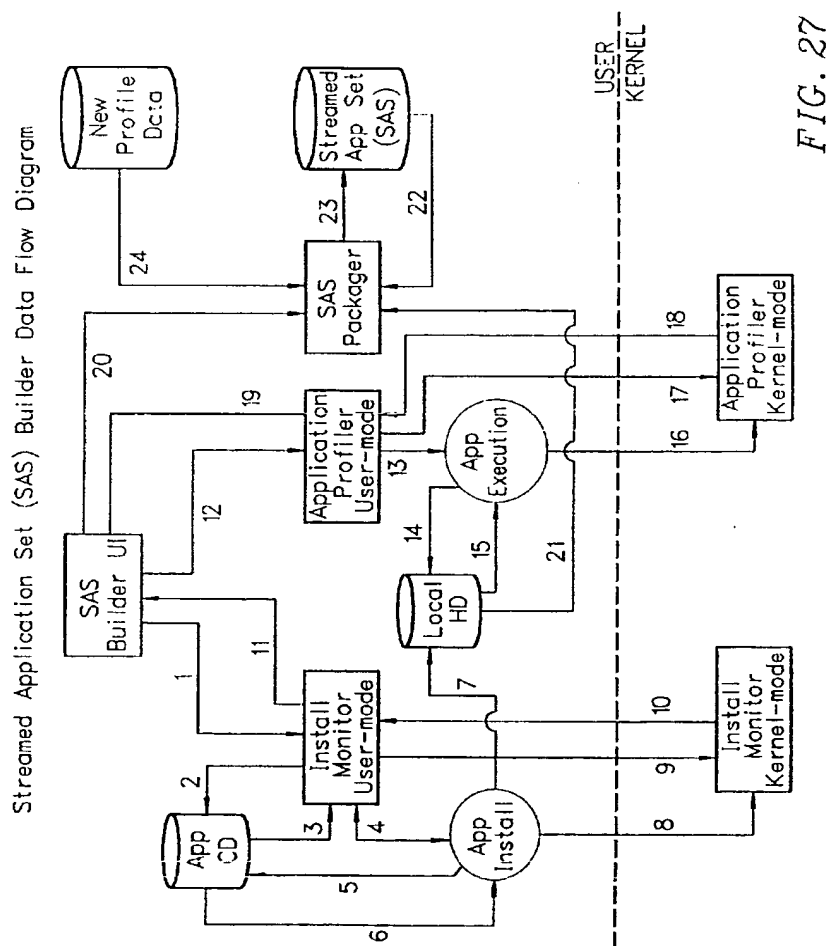


FIG. 27

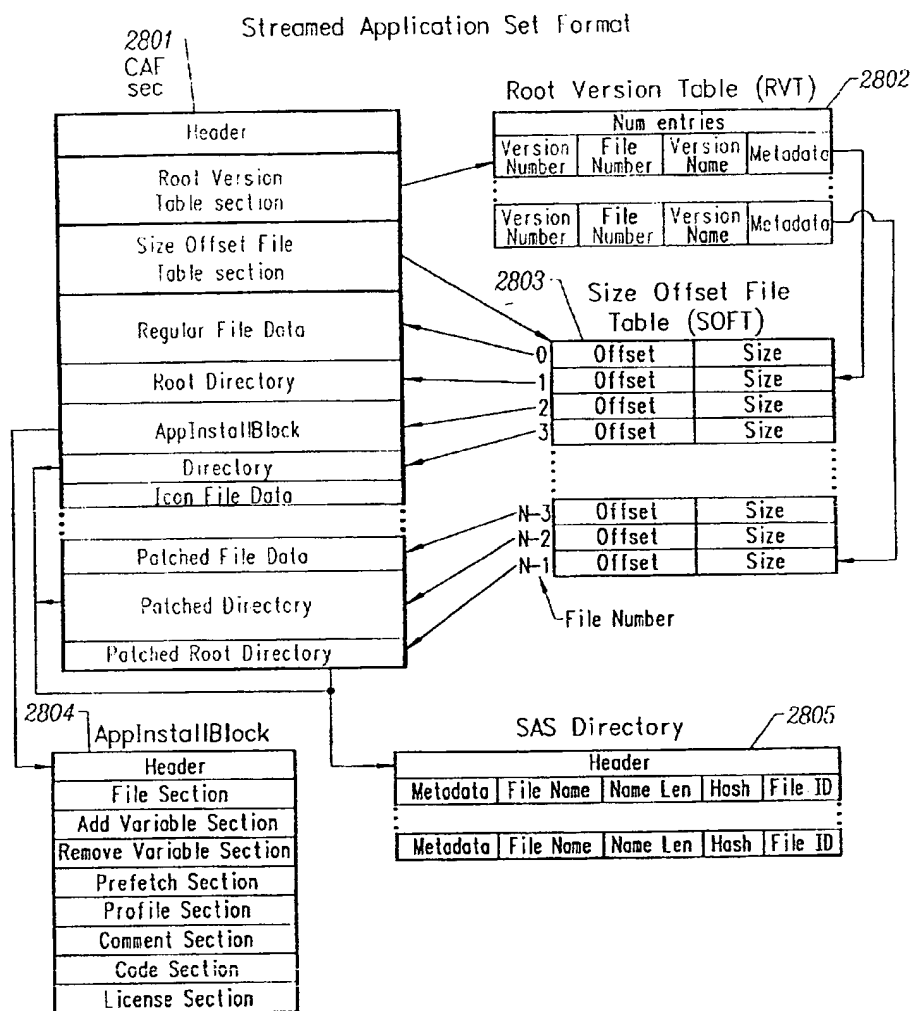


FIG. 28

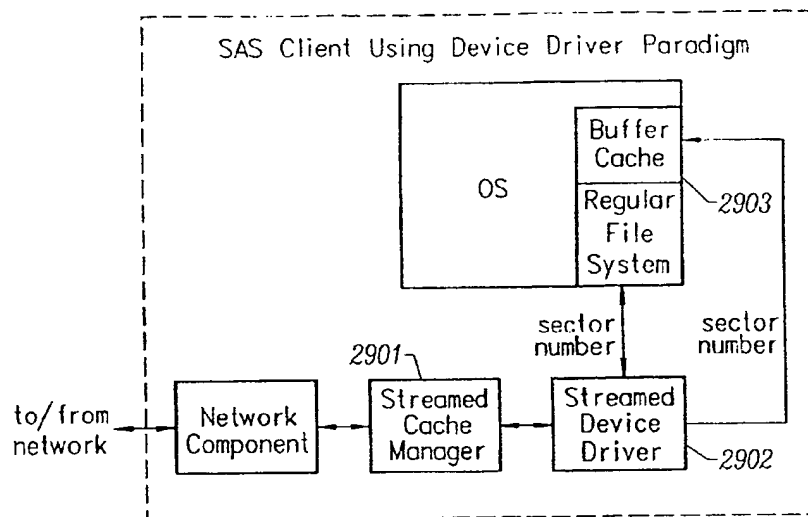


FIG. 29

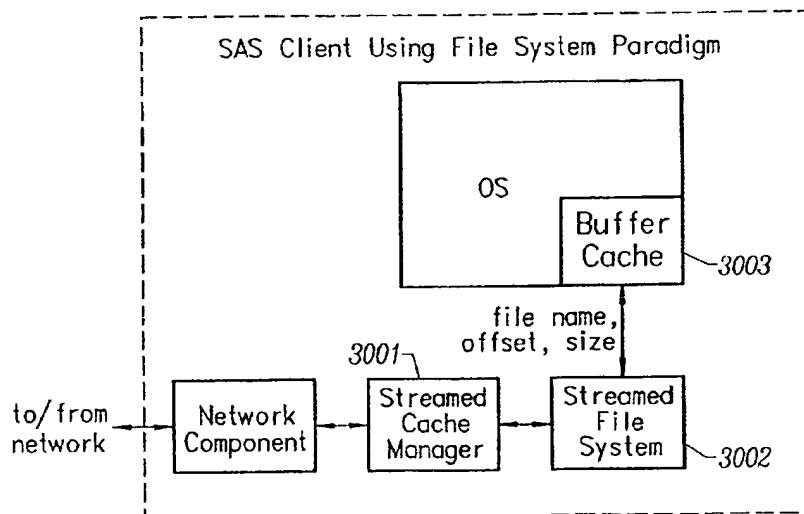


FIG. 30

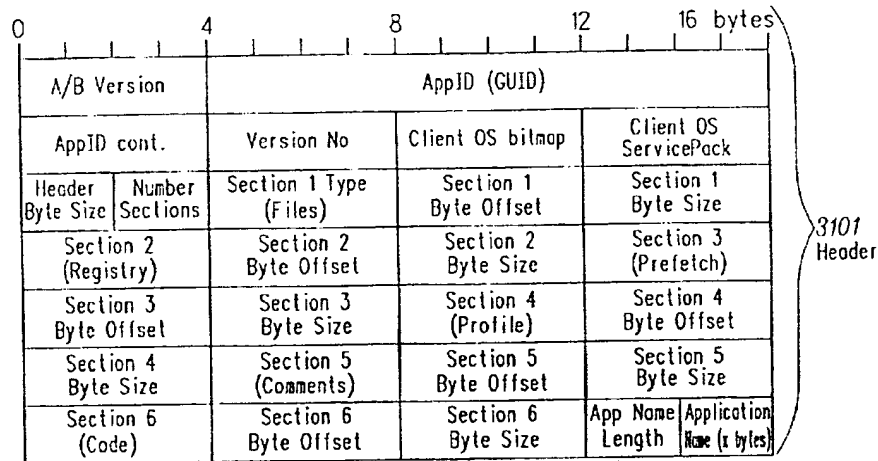


FIG. 31A

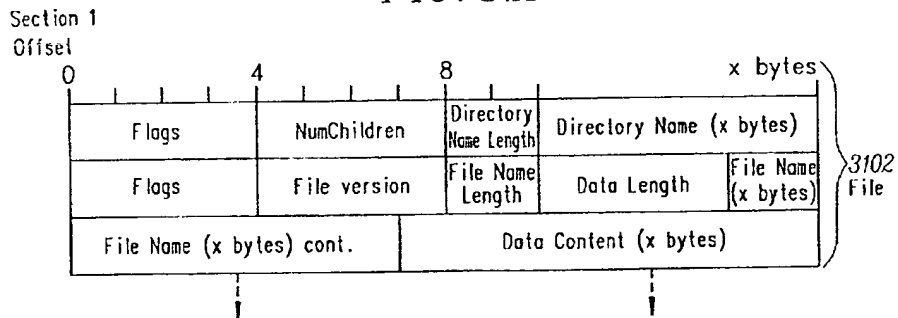


FIG. 31B

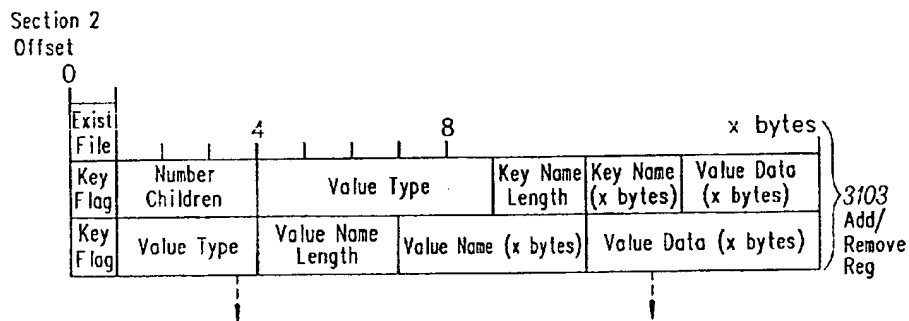


FIG. 31C

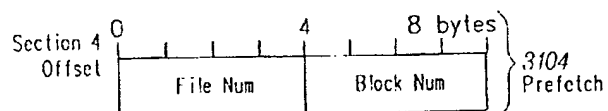


FIG. 31D

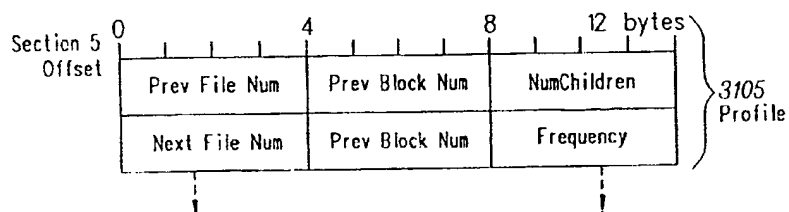


FIG. 31E

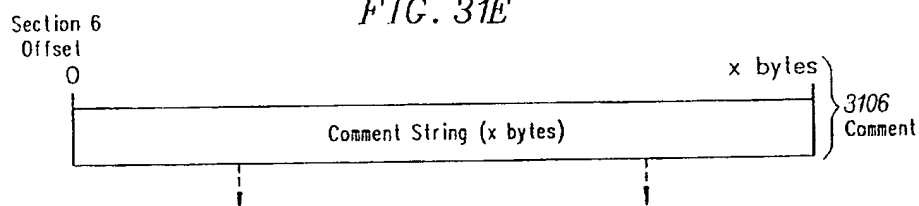


FIG. 31F

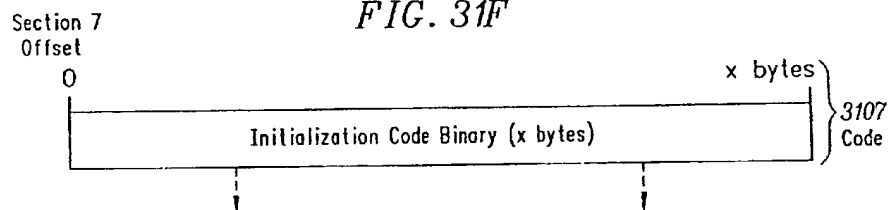


FIG. 31G

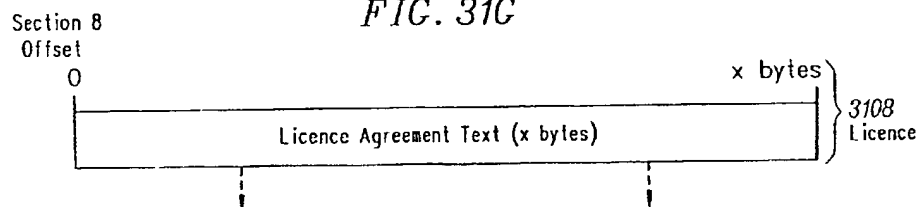


FIG. 31H

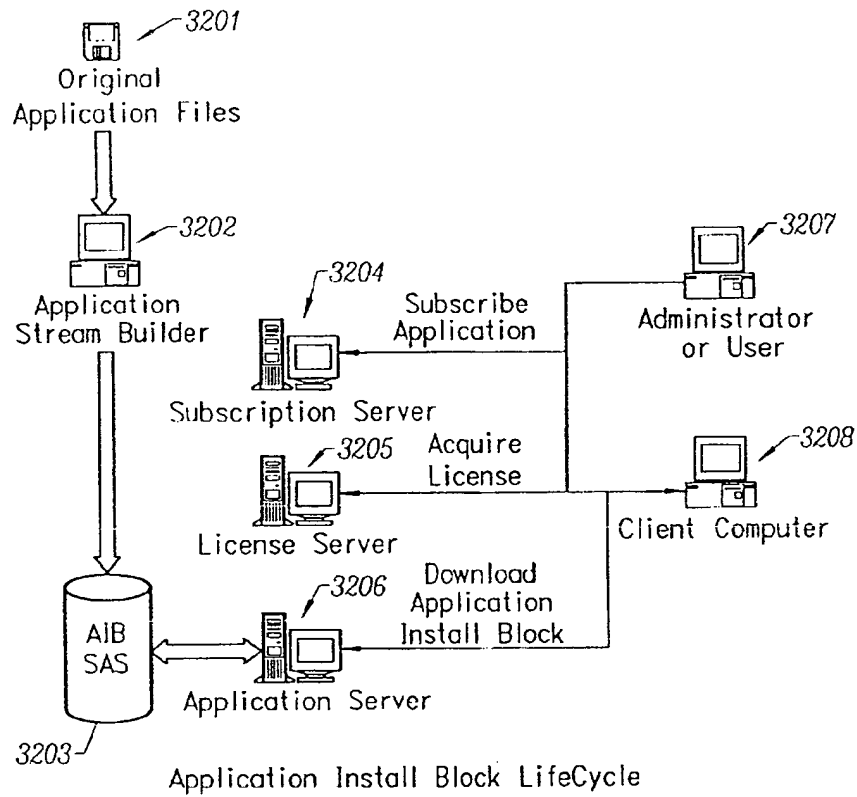


FIG. 32

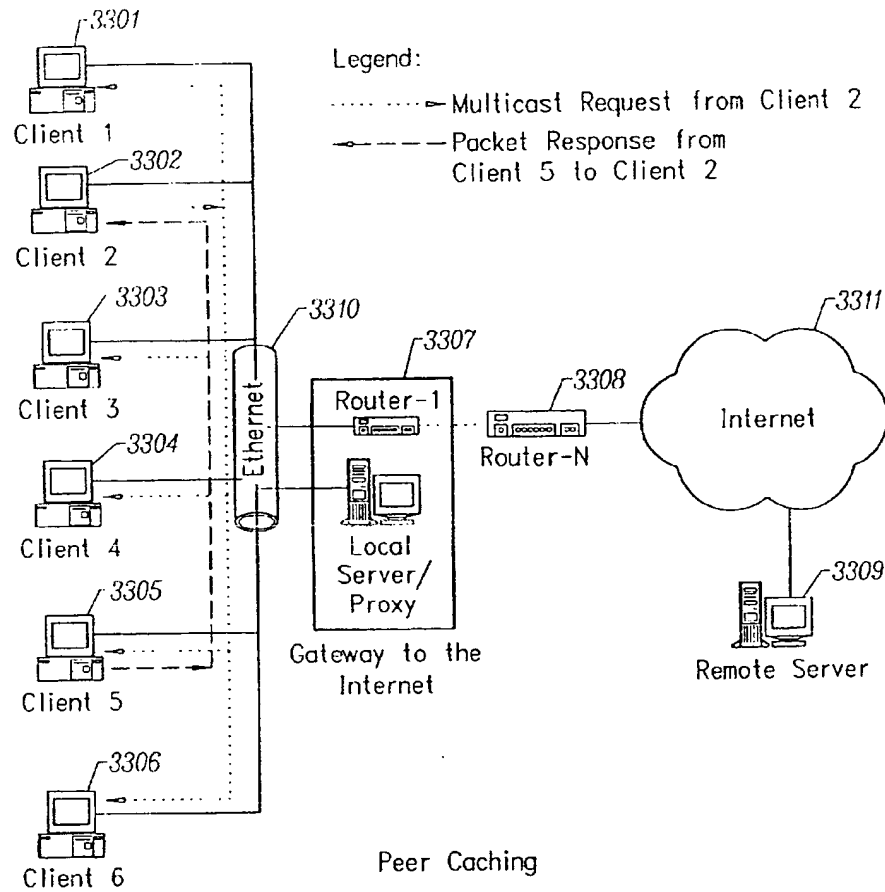
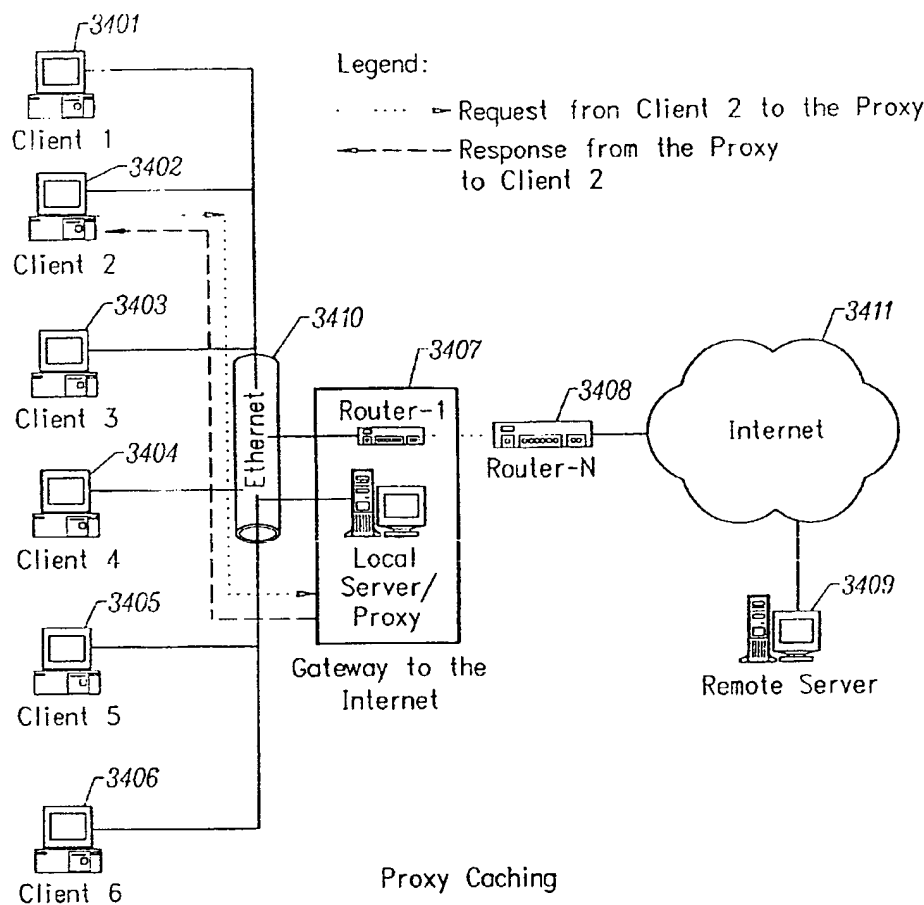


FIG. 33



Proxy Caching

FIG. 34

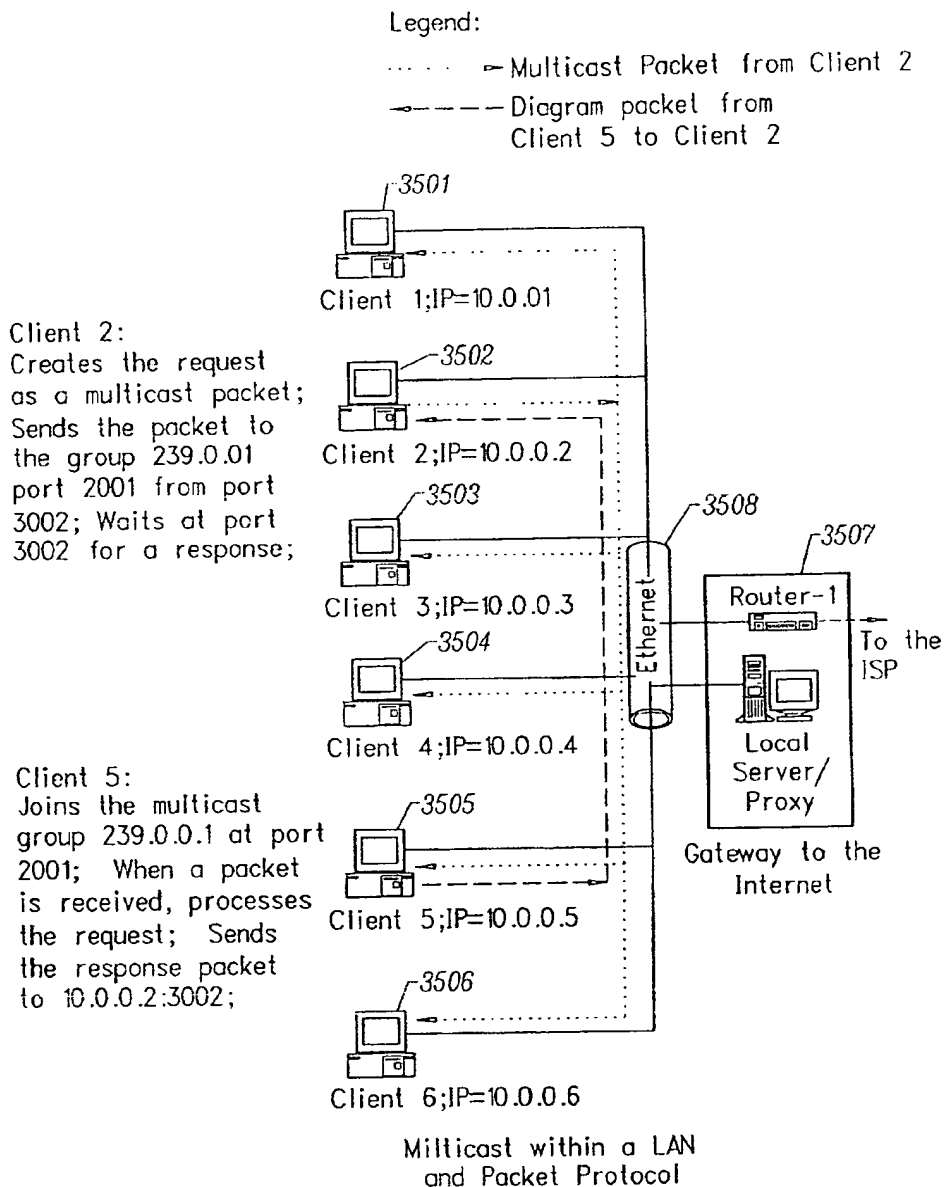
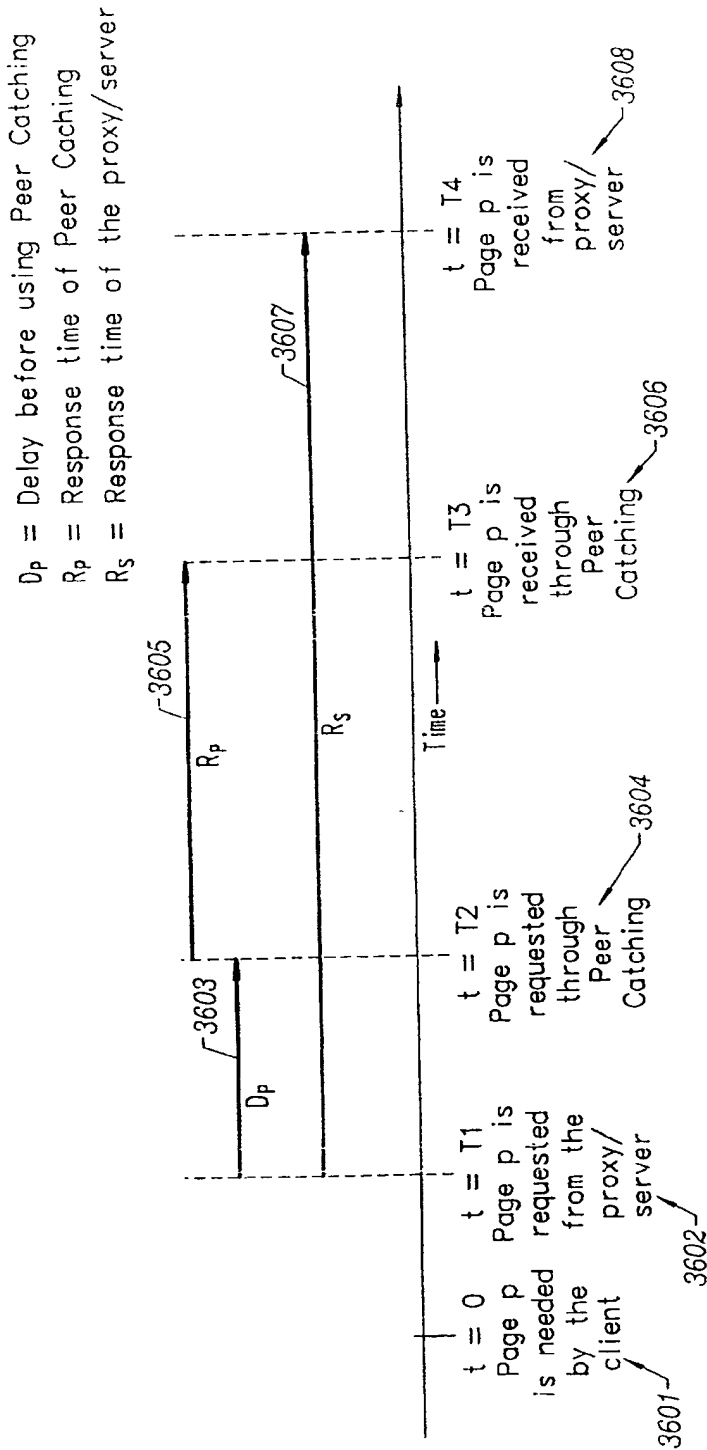
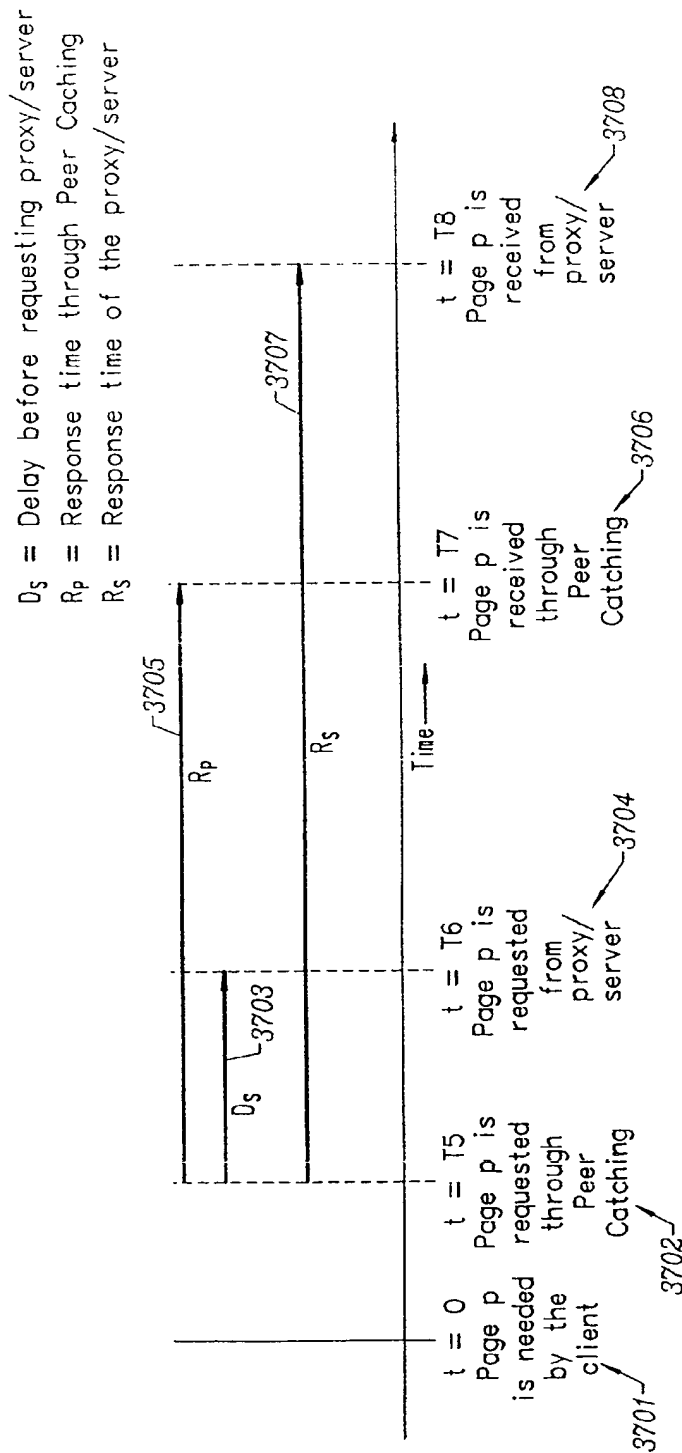


FIG. 35



Concurrent Requesting - Proxy First

FIG. 36



Concurrent Requesting - Peer Caching First

FIG. 37

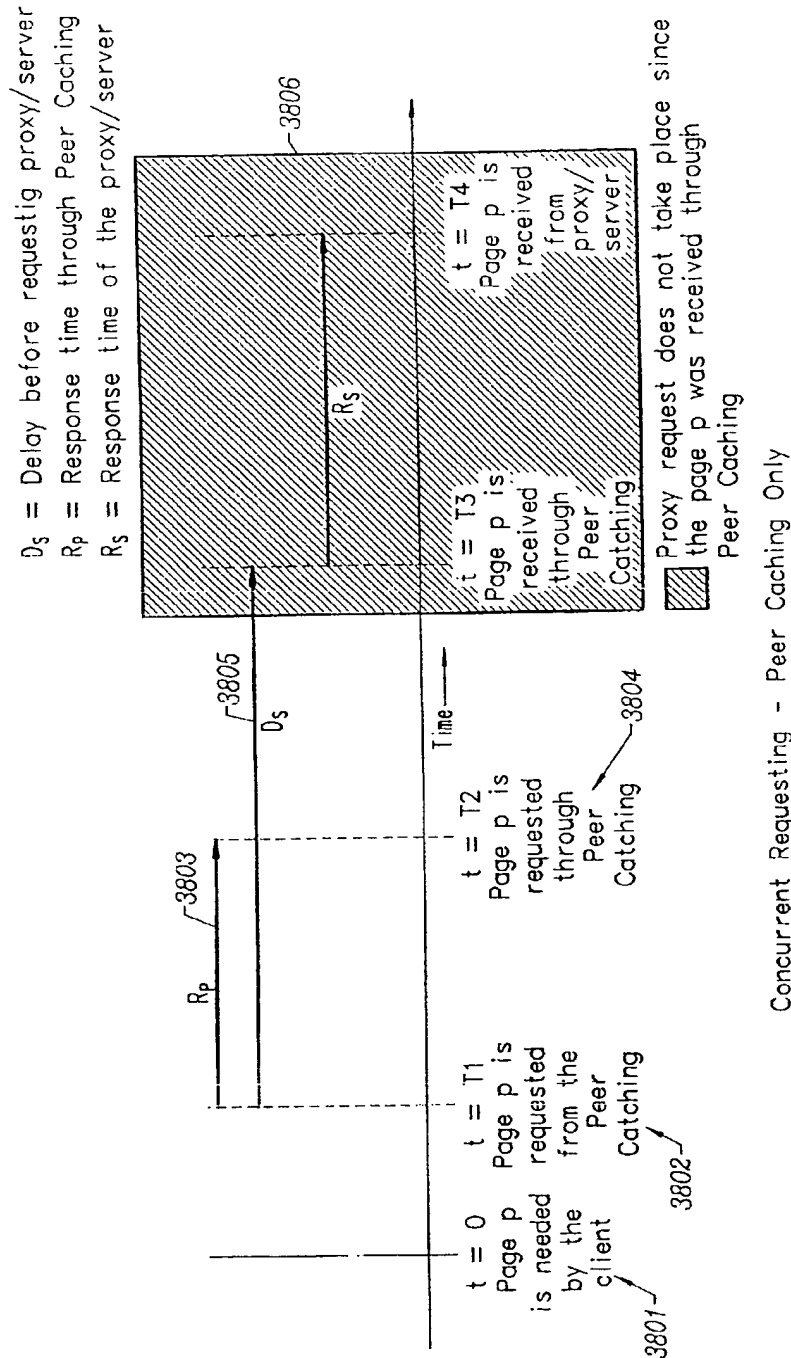
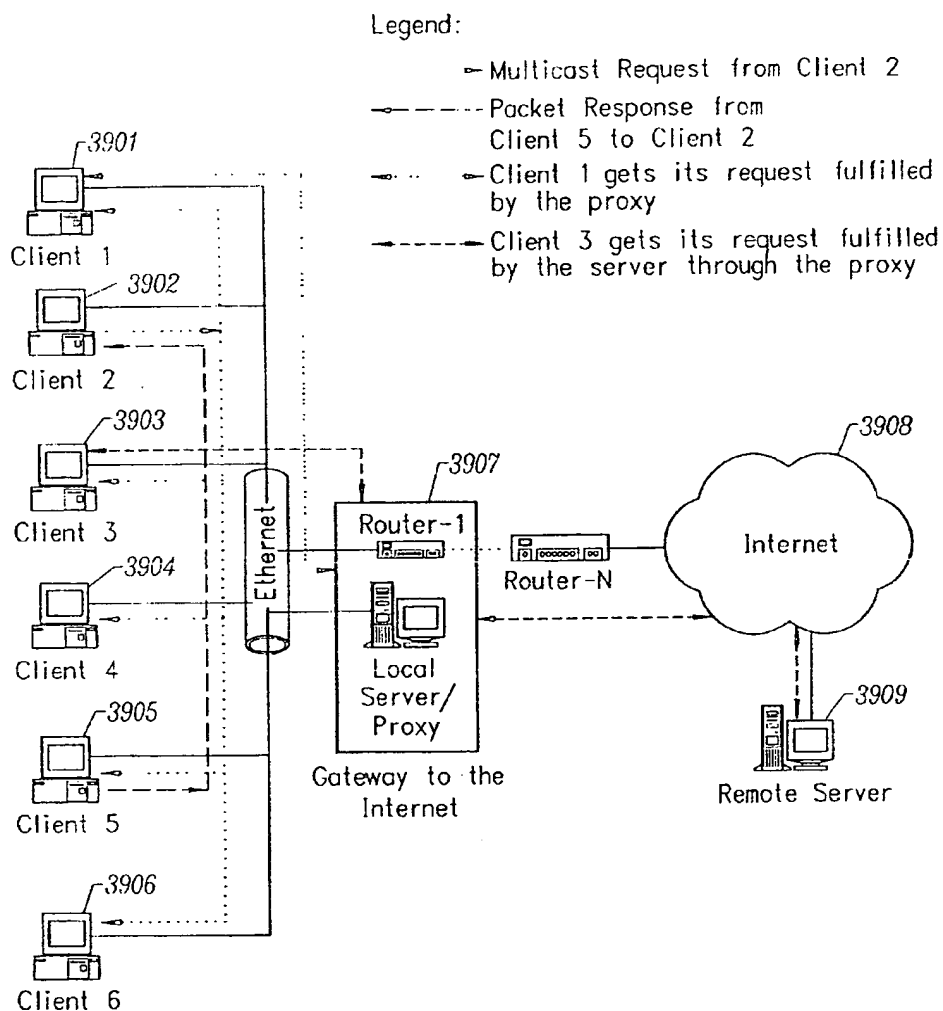


FIG. 38



Client-Server System with Peer and Proxy Caching

FIG. 39

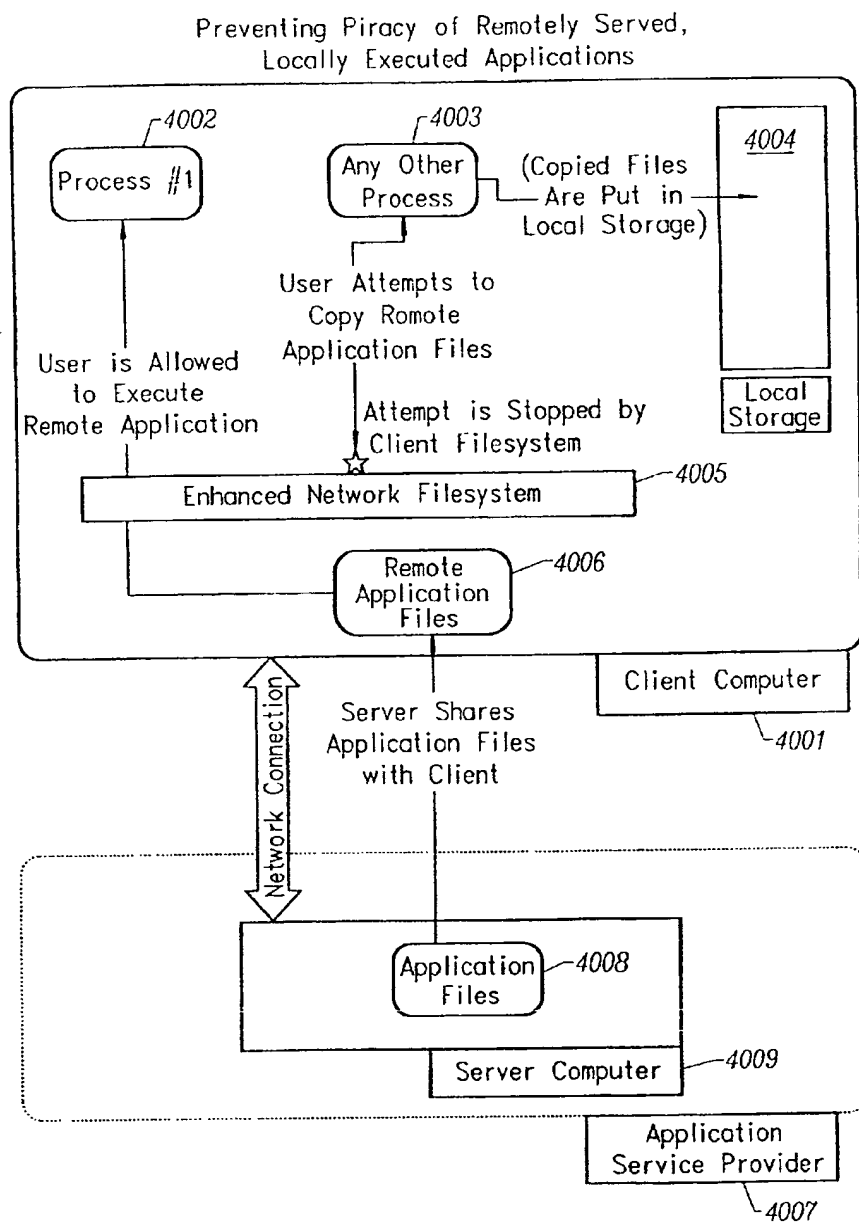


FIG. 40

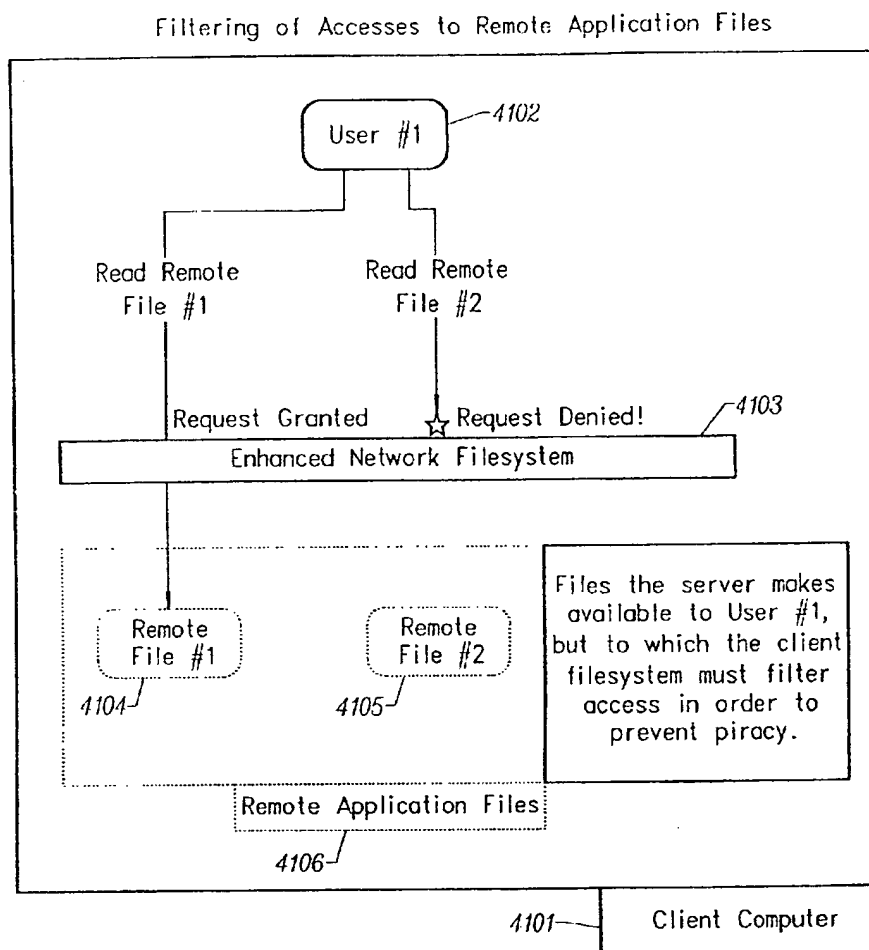


FIG. 41

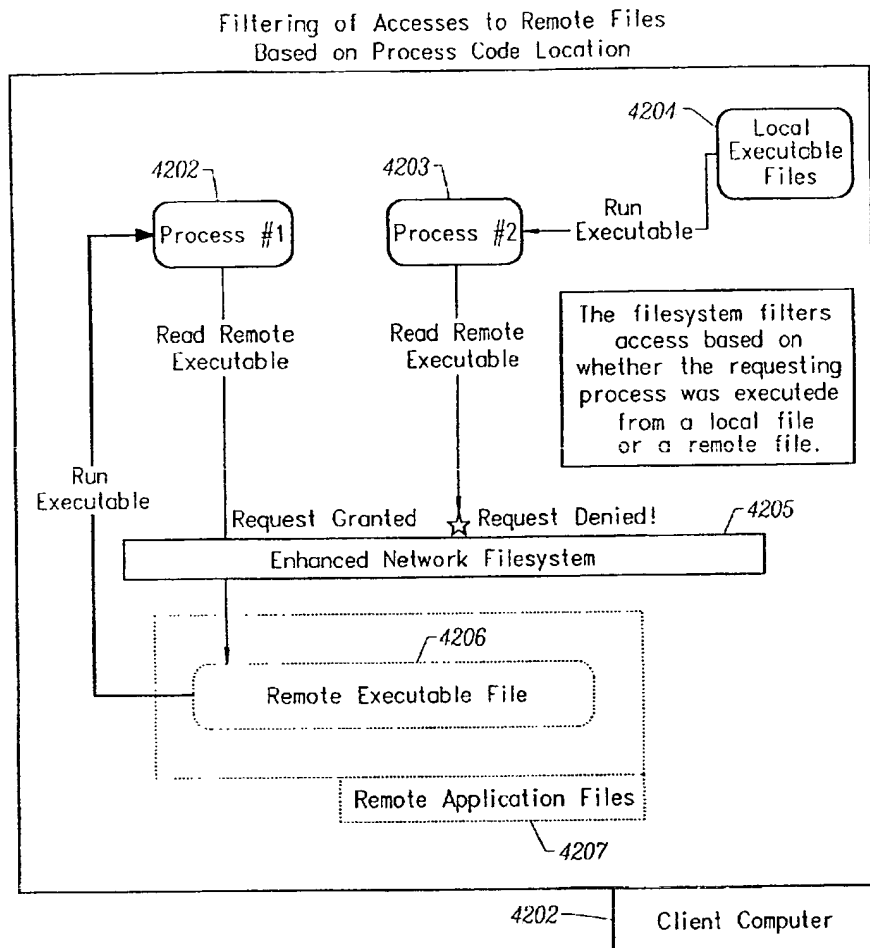


FIG. 42

Filtering of Accesses to Remote Files
Based on Targeted File Section

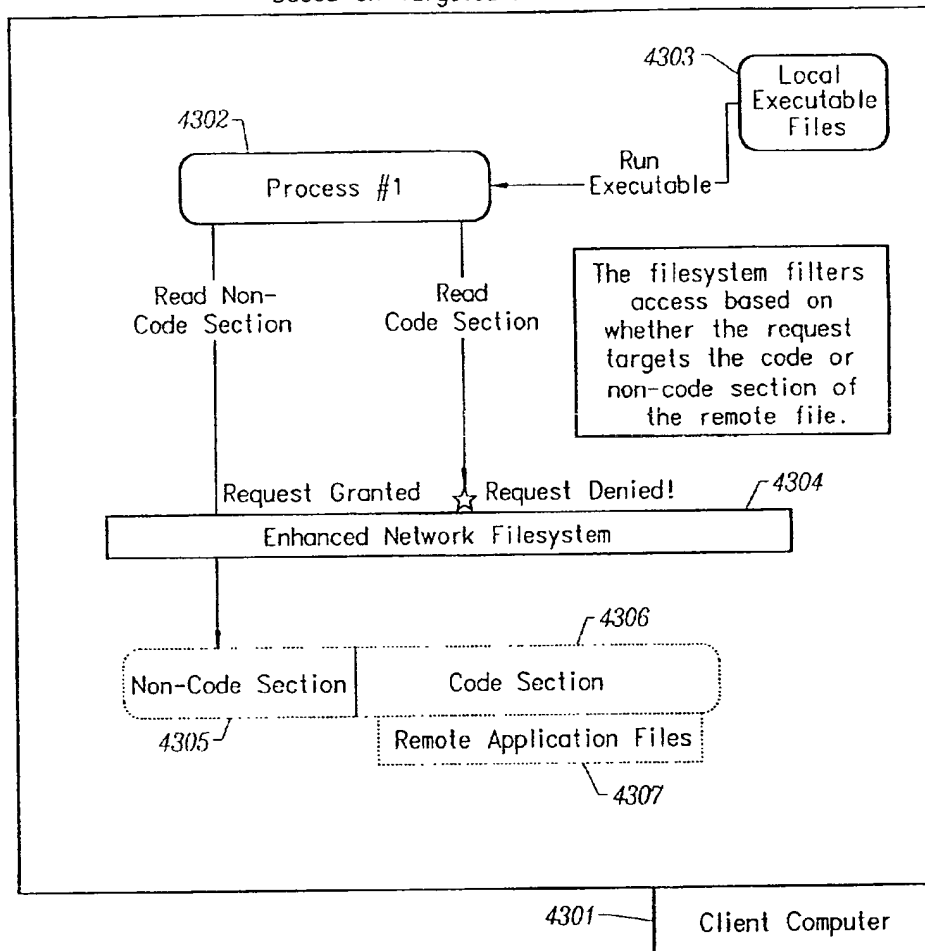


FIG. 43

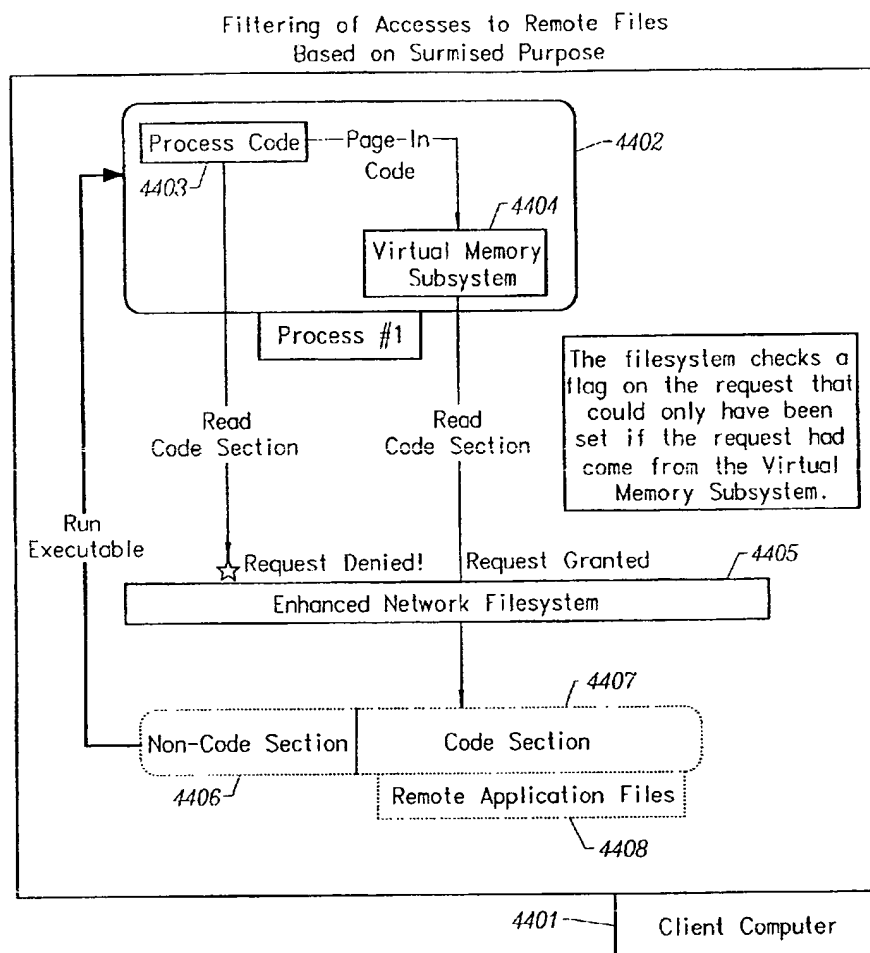


FIG. 44

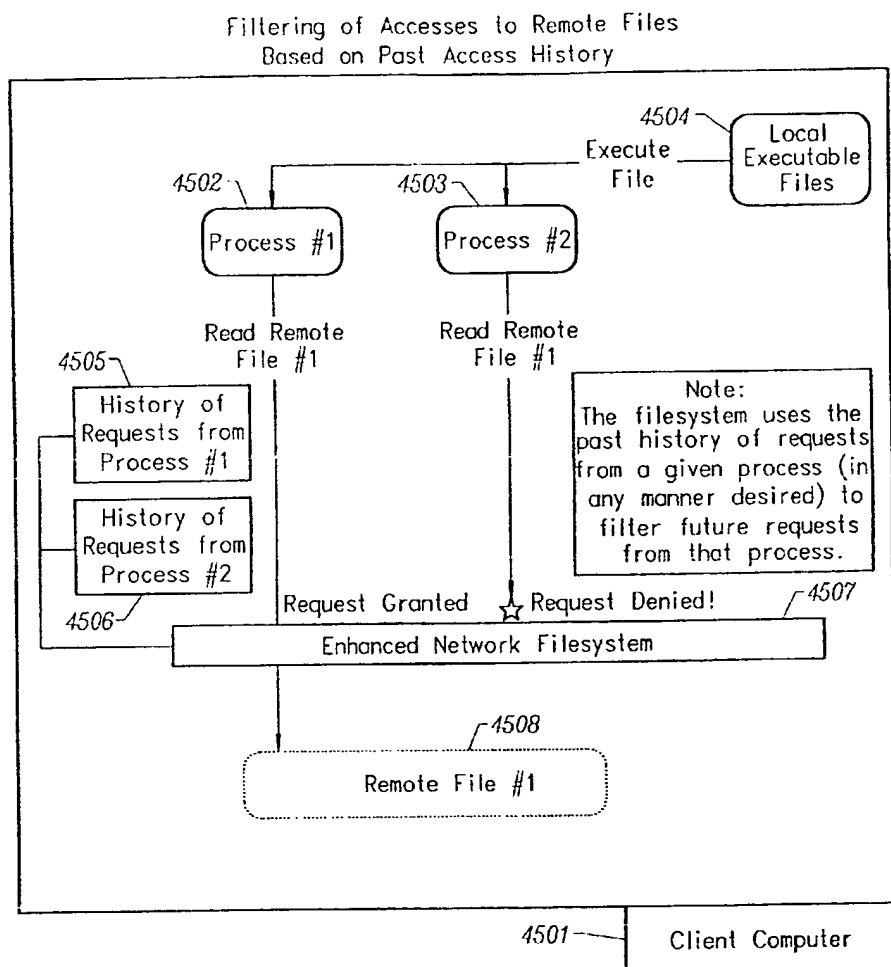


FIG. 45

CLIENT-SIDE PERFORMANCE OPTIMIZATION SYSTEM FOR STREAMED APPLICATIONS

CROSS REFERENCE TO RELATED APPLICATIONS

[0001] This application Claims benefit of U.S. Provisional Patent Application Ser. No. 60/246,384, filed on Nov. 6, 2000 (OTI.2000.0).

BACKGROUND OF THE INVENTION

[0002] 1. Technical Field

[0003] The invention relates to the streaming of computer program object code across a network in a computer environment. More particularly, the invention relates to client-side data retrieval, storage, and execution performance optimization techniques for computer program object code and other related data streamed across a network from a server.

[0004] 2. Description of the Prior Art

[0005] Retail sales models of computer application programs are fairly straight forward. The consumer either purchases the application program from a retailer that is either a brick and mortar or an ecommerce entity. The product is delivered to the consumer in a shrink-wrap form.

[0006] The consumer installs the program from a floppy disk or a CD-ROM included in the packaging. A serial number is generally provided that must be entered at installation or the first time the program is run. Other approaches require that the CD-ROM be present whenever the program is run. However, CD-ROMs are easily copied using common CDR technology.

[0007] Another approach is for the consumer to effectuate the purchase through an ecommerce entity. The application program is downloaded in its entirety to the consumer across the Internet. The consumer is emailed a serial number that is required to run the program. The consumer enters the serial number at the time the program is installed or the first time the program is run.

[0008] Once the application program is installed on a machine, it resides on the machine, occupying precious hard disk space, until it is physically removed. The installer portion of the program can also be installed on a server along with the installation files. Users within an intranet can install the program from the server, across the network, onto their machines. The program is a full installation of the program and resides on the user's machine until it is manually removed.

[0009] Trial versions of programs are also available online that are a partial or full installation of the application program. The program executes normally for a preset time period. At the end of the time period, the consumer is told that he must purchase the program and execution is terminated. The drawback to this approach is that there is an easy way for the consumer to fool the program. The consumer simply uninstalls the program and then reinstalls it, thereby restarting the time period.

[0010] Additionally, piracy problems arise once the application program is resident on the consumer's computer. Serial numbers for programs are easily obtained across the Internet. Software companies lose billions of dollars a year in revenue because of this type of piracy.

[0011] The above approaches fail to adequately protect software companies' revenue stream. These approaches also require the consumer to install a program that resides indefinitely on the consumer's hard disk, occupying valuable space even though the consumer may use the program infrequently.

[0012] The enterprise arena allows Application Service Providers (ASP) to provide browser-based implementations such as Tarantella offered by Santa Cruz Operation, Inc. in Santa Cruz, Calif. and Metaframe offered by Citrix Systems Inc. of Fort Lauderdale, Fla. A remote application portal site allows the user to click on an application in his browser to execute the application. The application runs on the portal site and GUI interfaces such as display, keystrokes and mouse clicks are transferred over the wire. The access to the program is password protected. This approach allows the provider to create an audit trail and to track the use of an application program.

[0013] AppStream Inc. of Palo Alto, Calif. uses Java code streamlets to provide streaming applications to the user. The system partitions a Web application program into Java streamlets. Java streamlets are then streamed to the user's computer on an as-needed basis. The application runs on the user's computer, but is accessed through the user's network browser.

[0014] The drawback to the browser-based approaches is that the user is forced to work within his network browser, thereby adding another layer of complexity. The browser or Java program manages the application program's run-time environment. The user loses the experience that the software manufacturer had originally intended for its product including features such as application invocation based on file extension associations.

[0015] It would be advantageous to provide a client-side performance optimization system for streamed applications that enables a client system to efficiently stream and execute application programs that are remotely served from a server. It would further be advantageous to provide a client-side performance optimization system for streamed applications that easily integrates into the client system's operating system.

SUMMARY OF THE INVENTION

[0016] The invention provides a client-side performance optimization system for streamed applications. The system enables a client system to efficiently stream and execute application programs that are remotely served from a server. In addition, the invention provides a system that easily integrates into the client system's operating system.

[0017] The invention provides several approaches for fulfilling client-side application code and data file requests for streamed applications. A streaming file system or file driver is installed on the client system that receives and fulfills application code and data requests.

[0018] One approach installs an application streaming file system on the client machine that appears to contain the installed application. The application streaming file system receive all requests for code or data that are part of the application and satisfies requests for application code or data by retrieving it from its persistent cache or by retrieving it directly from the streaming application server. Code or data retrieved from the server is placed in the cache for reuse.

[0019] Another approach installs a kernel-mode streaming file system driver and a user-mode client. Requests made to the streaming file system driver are directed to the user-mode client which handles the streams from the application streaming server or persistent cache and sends the results back to the driver.

[0020] Yet another approach is comprised of a streaming block driver on the client system. It appears as a physical disk to the native file system already installed on the client operating system. The driver receives requests for application code and data block reads and writes and satisfies the requests from the persistent cache or the streaming application server.

[0021] A final approach adds a disk driver and a user mode client on the client system. The disk driver sends program code and data requests to the user-mode client which satisfies them out of the persistent cache or by going to the streaming application server.

[0022] The persistent cache may be encrypted with a key not permanently stored on the client to prevent unauthorized use or duplication of application code or data. The key is sent to the client by the streaming application server upon application startup and is not stored in the application's persistent storage area.

[0023] The client can initiate the prefetching of application code and data to improve interactive application performance. The client software examines code and data requests and consults the contents of the persistent cache as well as historic information about application fetching patterns. It uses this information to request additional blocks of code and data from the streaming application server that it expects will be needed soon.

[0024] The server also initiates prefetching of application code and data by examining the patterns of requests made by the client and selectively returns to the client additional blocks that the client did not request but is likely to need soon.

[0025] A client-to-client communication mechanism allows local application customization to travel from one client machine to another without involving server communication.

[0026] A local copy-on-write file system allows some applications to write configuration or initialization files where they want to without rewriting the application, and without disturbing the local customization of other clients.

[0027] The invention disallows modifications to certain application files to prevent virus infections and reduce the chance of accidental application corruption. The system does not allow any data to be written to files that are marked as not modifiable. Attempts to mark the file as modifiable will not succeed.

[0028] The invention maintains checksums of application code and data and repairs damaged or deleted files by retrieving another copy from the application streaming server.

[0029] Applications are patched or upgraded via a change in the root directory for that application. The client can be notified of application upgrades by the streaming application server. The upgrades can be marked as mandatory, in which case the client will force the application to be upgraded.

[0030] The streaming application server broadcasts an application program's code and data and any client that is interested in that particular application program stores the broadcasted code and data for later use.

[0031] Other aspects and advantages of the invention will become apparent from the following detailed description in combination with the accompanying drawings, illustrating, by way of example, the principles of the invention.

BRIEF DESCRIPTION OF THE DRAWINGS

[0032] FIG. 1 is a block schematic diagram of a preferred embodiment of the invention showing components on the server that deal with users subscribing to and running applications according to the invention;

[0033] FIG. 2 is a block schematic diagram of a preferred embodiment of the invention showing the client components supporting application delivery and execution according to the invention;

[0034] FIG. 3 is a block schematic diagram of a preferred embodiment of the invention showing the components needed to install applications on the client according to the invention;

[0035] FIG. 4 is a block schematic diagram of the Builder that takes an existing application and extracts the Application File Pages for that application according to the invention;

[0036] FIG. 5a is a block schematic diagram illustrating how the Client Network Spoofer is used to handle mapping TCP interfaces to HTTP interfaces according to the invention;

[0037] FIG. 5b is a block schematic diagram illustrating how the Client Network Spoofer is used to handle mapping TCP interfaces to HTTP interfaces according to the invention;

[0038] FIG. 6a is a block schematic diagram showing several different components of the client software according to the invention;

[0039] FIG. 6b is a block schematic diagram showing the use of volatile and non-volatile storage of code and data in the client and server according to the invention;

[0040] FIG. 7a is a block schematic diagram showing one of two ways in which data may be compressed while in transit between the server and client according to the invention;

[0041] FIG. 7b is a block schematic diagram showing the other way in which data may be compressed while in transit between the server and client according to the invention;

[0042] FIG. 8 is a block schematic diagram showing an organization of the streaming client software according to the invention;

[0043] FIG. 9 is a block schematic diagram showing an alternative organization of the streaming client software according to the invention;

[0044] FIG. 10 is a block schematic diagram showing the application streaming software consisting of a streaming block driver according to the invention;

[0045] FIG. 11 is a block schematic diagram showing the application streaming software has been divided into a disk driver and a user mode client according to the invention;

[0046] FIG. 12 is a block schematic diagram showing the unencrypted and encrypted client caches according to the invention;

[0047] FIG. 13 is a block schematic diagram showing an application generating a sequence of code or data requests to the operating system according to the invention;

[0048] FIG. 14 is a block schematic diagram showing server-based prefetching according to the invention;

[0049] FIG. 15 is a block schematic diagram showing a client-to-client communication mechanism that allows local application customization to travel from one client machine to another without involving server communication according to the invention;

[0050] FIG. 16 is a block schematic diagram showing a client cache with extensions for supporting local file customization according to the invention;

[0051] FIG. 17 is a block schematic diagram showing aspects of a preferred embodiment of the invention related to load balancing and hardware fail over according to the invention;

[0052] FIG. 18 is a block schematic diagram showing the benefits to the use of compression in the streaming of Application File Pages according to the invention;

[0053] FIG. 19 is a block schematic diagram showing pre-compression of Application File Pages according to the invention;

[0054] FIG. 20 is a block schematic diagram showing multi-page compression of Application File Pages according to the invention;

[0055] FIG. 21 is a block schematic diagram showing profile-based prefetching according to the invention;

[0056] FIG. 22 is a block schematic diagram showing the use of tokens and a License Server according to the invention;

[0057] FIG. 23 is a block schematic diagram showing a flowchart for the Builder Install Monitor according to the invention;

[0058] FIG. 24 is a block schematic diagram showing a flowchart for the Builder Application Profiler according to the invention;

[0059] FIG. 25 is a block schematic diagram showing a flowchart for the Builder SAS Packager according to the invention;

[0060] FIG. 26a is a block schematic diagram showing versioning support according to the invention;

[0061] FIG. 26b is a block schematic diagram showing versioning support according to the invention;

[0062] FIG. 27 is a block schematic diagram showing a data flow diagram for the Streamed Application Set Builder according to the invention;

[0063] FIG. 28 is a block schematic diagram showing the Streamed Application Set format according to the invention;

[0064] FIG. 29 is a block schematic diagram showing an SAS client using a device driver paradigm according to the invention;

[0065] FIG. 30 is a block schematic diagram showing an SAS client using a file system paradigm according to the invention;

[0066] FIG. 31a through 31h is a schematic diagram showing various components of the AppinstallBlock format according to the invention;

[0067] FIG. 32 is a block schematic diagram showing the Application Install Block lifecycle according to the invention;

[0068] FIG. 33 is a block schematic diagram showing peer caching according to the invention;

[0069] FIG. 34 is a block schematic diagram showing proxy caching according to the invention;

[0070] FIG. 35 is a block schematic diagram showing multicast within a LAN and a packet protocol according to the invention;

[0071] FIG. 36 is a block schematic diagram showing concurrent requests for application server pages, for the case when the page is first requested through the proxy according to the invention;

[0072] FIG. 37 is a block schematic diagram showing concurrent requests for application server pages, for the case when the page is first requested through the peer caching according to the invention;

[0073] FIG. 38 is a block schematic diagram showing concurrent requests for application server pages, for the case when the page is received only through peer caching according to the invention;

[0074] FIG. 39 is a block schematic diagram showing a client-server system using peer and proxy caching according to the invention;

[0075] FIG. 40 is a block schematic diagram showing a preferred embodiment of the invention preventing piracy of remotely served, locally executed applications according to the invention;

[0076] FIG. 41 is a block schematic diagram showing the filtering of accesses to remote application files according to the invention;

[0077] FIG. 42 is a block schematic diagram showing the filtering of accesses to remote files based on process code location according to the invention;

[0078] FIG. 43 is a block schematic diagram showing the filtering of accesses to remote files based on targeted file section according to the invention;

[0079] FIG. 44 is a block schematic diagram showing the filtering of accesses to remote files based on surmised purpose according to the invention; and

[0080] FIG. 45 is a block schematic diagram showing the filtering of accesses to remote files based on past access history according to the invention.

DETAILED DESCRIPTION OF THE INVENTION

[0081] The invention is embodied in a client-side performance optimization system for streamed applications. A system according to the invention enables a client system to efficiently stream and execute application programs that are

remotely served from a server. In addition, the invention provides a system that easily integrates into the client system's operating system.

[0082] The invention provides a highly efficient and secure application delivery system in conjunction with the adaptively optimized execution of applications across a network such as the Internet, a corporate intranet, or a wide area network. This is done in such a way that existing applications do not need to be recompiled or recoded. Furthermore, the invention is a highly scalable, load-balancing, and fault-tolerant system that provides anti-piracy protection of the streamed applications.

[0083] When using the invention, an end-user requests applications that are resident on remote systems to be launched and run on the end-user's local system. The end-user's local system is called the client or client system, e.g., a desktop, laptop, palmtop, or information appliance. A remote system is called a server or server system and is located within a collection of one or more servers called a server cluster.

[0084] From the point of view of the client system, the application appears to be installed locally on the client even though it was initially installed on a different computer system. The applications execute locally on the client system and not on the server system. To achieve this result, the application is converted into a form suitable for streaming over the network. The streaming-enabled form of an application is called the Streamed Application Set (SAS) and the conversion process is termed the SAS Builder. The conversion of an application into its SAS form typically takes place on a system different from either an end-user client system or an Application Service Provider Server Cluster. This system is called the SAS Conversion System or, simply, the conversion system.

[0085] Components of the invention are installed on the client system to support activities such as the installation, invocation, and execution of a SAS-based application. Other components of the invention are installed on the server system to support activities such as the verification of end user application subscription and license data and the transfer and execution of a SAS-based application on the client system. Some of the client and some of the server components run in the kernel-mode while other components run in the usual user-mode.

[0086] The term Application Service Provider (ASP) refers to an entity that uses the server components on one or more server systems, i.e., an ASP Server Cluster, to deliver applications to end-user client systems. Such an entity could be, for example, a software manufacturer, an e-commerce vendor that rents or leases software, or a service department within a company. The invention enables an ASP to deliver applications across a network, in a highly efficient and secure way; the applications are adaptively optimized for execution on an end-user's client system.

[0087] A number of techniques are employed to increase the overall performance of the delivery of an application and its subsequent execution by minimizing the effect of network latency and bandwidth. Among the techniques employed are: the SAS Builder identifies sequences of frequently accessed application pages and uses this information when generating a SAS; individual SAS pages and

sequences of SAS pages are compressed and cached in an in-memory cache on the server system; various aspects of the applications are monitored during their actual use on a client and the resulting profiling data is used by the client to pre-fetch (pull) and by the server to send (push) additional pages which have a high likelihood of being used prior to their actual use; and SAS pages are cached locally on a client for their immediate use when an application is invoked.

[0088] Aggregate profile data for an application, obtained by combining the profile data from all the end-user client systems running the application, is used to increase the system performance as well. A number of additional caching techniques that improve both system scalability and performance are also employed. The above techniques are collectively referred to as collaborative caching.

[0089] In an embodiment of the invention, the SAS Builder consists of three phases: installation monitoring, execution profiling, and application stream packaging. In the final SAS Builder phase, the Application Stream Packager takes the information gathered by the Application Install Monitor and the Application Execution Profiler and creates the SAS form of the application, which consists of a Stream Enabled Application Pages File and a Stream Enabled Application Install Block.

[0090] The Stream Enabled Application Install Block is used to install a SAS-based application on a client system while selected portions of the Stream Enabled Application Pages File are streamed to a client to be run on the client system. The Stream Enabled Application Install Block is the first set of data to be streamed from the server to the client and contains, among other things, the information needed by the client system to prepare for the streaming and execution of the particular application. Individual and aggregate client dynamic profile data is merged into the existing Stream Enabled Application Install Block on the server to optimize subsequent streaming of the application.

[0091] The invention employs a Client Streaming File System that is used to manage specific application-related file accesses during the execution of an application. For example, there are certain shared library files, e.g., "foo.dll", that need to be installed on the local file system, e.g., "c:\winnt\system32\foo.dll", for the application to execute. Such file names get added to a "spoof database". For the previous example, the spoof database would contain an entry saying that "c:\winnt\system32\foo.dll" is mapped to "z:\word\winnt\system32\foo.dll" where "z:" implies that this file is accessed by the Client Streaming File System. The Client Spoofer will then redirect all accesses to "c:\winnt\system32\foo.dll" to "z:\word\winnt\system32\foo.dll". In this manner, the client system gets the effect of the file being on the local machine whereas in reality the file is streamed from the server. Several different classes of files can be treated in this way, e.g., specific application registry entries and application-based networking calls when such calls cross a firewall.

[0092] Lastly, the invention incorporates a number of software anti-piracy techniques directed at combating the piracy of applications of the type described herein that are delivered to the end-user over a network for execution on a client system. Among the anti-piracy techniques included are: client-side fine-grained filtering of file accesses directed at remotely served files; filtering of file accesses based on

where the code for the process that originated the request is stored; identification of crucial portions of application files and filtering file access depending on the portions of the application targeted; filtering of file accesses based on the surmised purpose of the file access, as determined by examining the program stack or flags associated with the request; and filtering of file accesses based on the surmised purpose of the file access, as determined by examining a history of previous file accesses by the same process.

[0093] As mentioned above, the invention provides server and client technology for streaming application delivery and execution. The invention includes secure license-based streaming delivery of applications over Internet/extranets/intranets utilizing client-based execution with client caching and server-based file accesses by page.

[0094] 1. The invention provides many advantages over the present approaches, including:

[0095] Secure license-based streaming delivery over Internet/extranets/intranets:

[0096] reduces IT costs over client installation;

[0097] supports rental model of app delivery, which opens new markets and increases user convenience over purchase and client installation; and

[0098] enhances the opportunities to prevent software piracy over purchase and client installation.

[0099] Client-based execution with client caching:

[0100] increases typical application performance over server-based execution;

[0101] reduces network latency and bandwidth usage over non-cached client execution; and

[0102] allows existing applications to be run w/o rewrite/recompile/rebuild unlike other explicitly-distributed client/server application delivery approaches.

[0103] Server-based file accesses:

[0104] improve server-scaling over server-based execution;

[0105] allow transparent failover to another server whereas server-based execution does not;

[0106] make server load balancing easier than it is with server-based execution; and

[0107] allow increased flexibility in server platform selection over server-based execution.

[0108] Server-based file accesses by page:

[0109] reduce network latency over complete file downloads;

[0110] reduce network bandwidth overhead over complete file downloads; and

[0111] reduce client cache footprint over complete file downloads.

[0112] 2. Features of the Invention

[0113] A) Server Components Supporting Application Delivery and Execution.

[0114] i) referring to FIG. 1, the server components include:

[0115] a. Client/server network interface 110 that is common to the client 113 and the server. This is the communication mechanism through which the client and the server communicate.

[0116] b. The Subscription Server 105—This is the server the client 113 connects to for subscribing and unsubscribing applications. This server then adds/deletes the subscription information to the Subscription Database 101 and also updates the License Database 102 with the information stating that the client 113 can/cannot run the subscribed information under the agreed upon licensing terms. This communication between the client 113 and the Subscription Server 105 happens over SSL that is an industry standard protocol for secure communication. The Subscription Server 105 is also contacted for updating any existing subscription information that is in the Subscription Database 101.

[0117] c. The License Server 106—This is the server the client 113 connects to for getting a license to run an application after the client has subscribed to the application. This server validates the user and his subscriptions by consulting the License Database 102. If the client 113 does have a valid license, the License Server 106 sends an "Access token" to the client that is encrypted using an "encryption key" that the License Database 102 obtains from the Server Config Database 103. The "Access token" contains information like the Application ID and an expiration time. Along with the "Access token," the License Server 106 also sends a list of least loaded application servers that it obtains from the Server Config Database 103 and also the expiration time that was encoded in the "Access token". The client 113 uses this expiration time to know when to ask for a new token. This communication between the client 113 and the License Server 106 happens over SSL.

[0118] d. The Application Server 107—Once the client 113 obtains an "Access token" to run an application, it connects to the Application Server 107 and presents to it the "Access token" along with the request for the application bits. Note that the "Access token" is opaque to the client 113 since it does not have the key to decrypt it. The Application Server 107 validates the "Access token" by decrypting it using a "decryption key" obtained from the Server Config Database 103 and checking the content against a predefined value like for example the Application ID and also by making sure that the expiration time in the "Access token" has not elapsed. It then serves the appropriate bits to the client 113 to enable it to run the application. The encryption and decryption keys could be something like a private key/public key pair or a symmetric key or any other means of providing security. Note that the keys are uniform across all the servers within an ASP.

- [0119] e. The Monitor Server 108—It monitors the load in terms of percent of CPU utilization on the Application Servers 107 and the License Servers 106 on a periodic basis (for example—every minute) and adds that information to the Server Config Database 103.
- [0120] f. The Profile Server 109—It receives profile information from the clients periodically. It adds this information to the Profile Database 104. The Profile Server 109 based on the profile information from different clients updates the App Prefetch Page List section of the Stream App Install Blocks 112.
- [0121] ii) The data structures supporting the above server components include:
- [0122] a. Subscription Database 101—This is the database that stores the user information in terms of username, list of apps subscribed, password, billing information, address, group, admin. The username is the primary key. The admin field identifies if this user has admin privileges for the group he belongs to.
- [0123] b. License Database 102—This is the database that stores licensing information, i.e., which user can run what application and under which license. This database also keeps track of usage information, i.e., which user has used which application for how long and how many times. The information looks like:
- [0124] Username, application, time of usage, number of times run
- [0125] Username, application, licensing policy
- [0126] Username, application, is app running, no of instances, time of start The username is the primary key. The licensing policy could be something simple like expiry date or something more complicated like number of instances simultaneously allowed within a group etc.
- [0127] c. Server Config Database 103—This database stores information about which server can run which application, what is the load on all the servers, what is the encryption “key” to be used by the servers and all other information that is needed by the servers. The information looks like:
- [0128] Server IP address, App/Slim server, application list, current load
- [0129] Encryption key, Decryption key
- [0130] The Server IP address is the primary key for the first table. The keys are common across all servers.
- [0131] d. Profile Database 104—This database stores the profile information received by the profile server from the clients periodically. The information looks like:
- [0132] Application ID, File ID, Block ID number of bits
- [0133] The Application ID is the primary key.
- [0134] e. Application File Pages 111—This is the one of the outputs of the “builder” as explained below and is put on the Application Server 107 so that it can serve the appropriate bits to the client.
- [0135] f. Stream App Install Blocks 112—This is the other output of the “builder” and contains the information for successfully installing applications on the client for streaming applications.
- [0136] B) Client Components Supporting Application Delivery & Execution
- [0137] i) With respect to FIGS. 1 and 2, these client components include:
- [0138] a. Client/Server Network interface 202—This is the same interface as explained above.
- [0139] b. Client License Manager 205—This component requests licenses (“Access tokens”) from the License Server 106 when the client wants to run applications. The License Server 106 sends an “Access token” to the client that can be used to run the applications by presenting it to the Application Server 107. Along with the token, the License Server 106 also sends the expiry time of the token. The Client License Manager 205 renews the token just before the expiry period so that the client can continue running the application. When the application is complete, the Client License Manager 205 releases the token by sending a message to the License Server 106. In addition, when a user has subscribed to an application, the Client License Manager 205 first checks to make sure that the application is installed on the machine the user is trying to run the application from and if not requests for the application installation. It does this using a list of Installed Apps that it maintains.
- [0140] c. Client Cache Manager 207—This component caches the application bits received from the Application Server 107 so that next time a request is made to the same bits, the request can be served by the cache instead of having to go to the Application Server 107. The Client Cache Manager 207 has a limited amount of space on the disk of the client machine that it uses for the cache. When the space is fully occupied, the Client Cache Manager 207 uses a policy to replace existing portions of the cache. This policy can be something like LRU, FIFO, random etc. The Client Cache Manager 207 is responsible for getting the application bits requested by the Client Streaming File System 212. If it does not have the bits cached, it gets them from the Application Server 107 through the network interface. However it also need to get the “Access token” from the Client License Manager 205 that it needs to send along with the request for the application bits. The Client Cache Manager 207 also updates the Prefetch History Info 209 with the requests it receives from the Client Streaming File System 212.
- [0141] d. Client Streaming File System 212—This component serves all file system requests made by the application running on the client. The application makes calls like “read”, “write” etc. to files that need to be streamed. These requests lead to page faults in the operating system and the page faults are handled by the Client Streaming File System 212 that in turn asks the Client Cache Manager 207 for the appropriate bits. The Client Cache Manager 207 will send

those bits from the cache if they exist there or forward the request to the Application Server 107 through the network interface to get the appropriate bits.

[0142] e. Client Prefetcher 208—This component monitors the requests made by the client to the Application Server 107 and uses heuristics to make additional requests to the Application Server 107 so that the bits can be obtained from the Application Server 107 before the client machine makes the request for them. This is mainly to hide the latency between the client and the Application Server 107. The history information of the requests is stored in the Prefetch History Info file 209.

[0143] f. Client Profiler 203—At specific time intervals, the client profiler sends the profile information, which is the Prefetch History Info to the prefetch server at the ASP that can then update the App Prefetch Page Lists for the different applications accordingly.

[0144] g. Client File Spoofer 211—Certain files on the client need to be installed at specific locations on the client system. To be able to stream these files from the Application Server 107, the Client Spoofer 211 intercepts all requests to these files made by a running application and redirects them to the Client Streaming File System 212 so that the bits can be streamed from the Application Server 107.

[0145] h. Client Registry Spoofer 213—Similar to files, certain registry entries need to be different when the application being streamed is running and since it is undesirable to overwrite the existing registry value, the read of the registry value is redirected to the Client Registry Spoofer 213 which returns the right value. However, this is optional as it is very likely that overwriting the existing registry value will make the system work just fine.

[0146] i. Client Network Spoofer 213—Certain applications make networking calls through a protocol like TCP. To make these applications work across firewalls, these networking calls need to be redirected to the Client Network Spoofer 213 which can tunnel these requests through a protocol like HTTP that works through firewalls.

[0147] ii) The data structures needed to support the above client components include:

[0148] a. File Spoof Database 210—The list of files the requests to which need to be redirected to the Client Streaming File System 212. This information looks like (The source file name is the primary key)

[0149] Source File Name, Target File Name

[0150] b. Registry Spoof Database 216—List of registry entries and their corresponding values that need to be spoofed. Each entry looks like:

[0151] Registry entry, new value

[0152] c. Network Spoof Database 214—Like of IP addresses, the networking connections to which need to be redirected to the Client Network Spoofer 213. Each entry looks like (IP address is the primary key):

[0153] IP address, Port number, new IP address, new Port number

[0154] d. Client Stream Cache 206—The on-disk cache that persistently stores application bits.

[0155] e. Known ASPs and Installed Apps 204—The list of ASP servers (Application, License and Subscription) and also the list of applications that are installed on the client.

[0156] f. Prefetch History Info 209—The history of the requests made to the cache. This consists of which blocks were requested from which file for which application and how many times each block was requested. It also consists of predecessor-successor information indicating which block got requested after a particular block was requested.

[0157] C) Client Application Installation

[0158] Referring to FIG. 3, the client application installation components include:

[0159] i) Client License Manager 303—This is the same component explained above.

[0160] ii) Client Application Installer 305—This component is invoked when the application needs to be installed. The Client Application Installer 305 sends a specific request to the Application Server 107 for getting the Stream App Install Block 301 for the particular application that needs to be installed. The Stream App Install Block 301 consists of the App Prefetch Page List 306, Spoof Database 308, 309, 310, and App Registry Info 307. The Client Application Installer 305 then updates the various Spoof Databases 308, 309, 310 and the Registry 307 with this information. It also asks the Client Prefetcher 208 to start fetching pages in the App Prefetch Page List 306 from the Application Server 107. These are the pages that are known to be needed by a majority of the users when they run this application.

[0161] D) Application Stream Builder Input/Output

[0162] With respect to FIG. 4, the Builder components include the following:

[0163] i) Application Install Monitor 403—This component monitors the installation of an application 401 and figures out all the files that have been created during installation 402, registry entries that were created and all the other changes made to the system during installation.

[0164] ii) Application Profiler 407—After the application is installed, it is executed using a sample script. The Application Profiler 407 monitors the application execution 408 and figures out the application pages that got referenced during the execution.

[0165] iii) App Stream Packager 404—The App Stream Packager 404 takes the information gathered by the Application Profiler 407 and the Application Install Monitor 403 and forms the Application File Pages 406 and the Stream App Install Block 405 from that information.

[0166] E) Network Spoofing for client-server applications:

[0167] Referring to FIGS. 1, 4, 5a, 5b, and 6a, the component that does the Network Spoofing is the TCP to HTTP converter 503, 507. The basic idea is to take TCP packets and tunnel them through HTTP on one side and do exactly the opposite on the other. As far as the client 501 and the server 502 are concerned the communication is TCP and so existing applications that run with that assumption work unmodified. This is explained in more detail below.

[0168] On the client side, the user launches an application that resides on the Client Streaming File System. That application may be started in the same ways that applications on other client file systems may be started, e.g., opening a data file associated with the application or selecting the application from the Start/Programs menu in a Windows system. From the point of view of the client's operating system and from the point of view of the application itself, that application is located locally on the client.

[0169] Whenever a page fault occurs on behalf of any application file residing on the Client Streaming File System 604, that file system requests the page from the Client Cache Manager 606. The Client Cache Manager 606, after ensuring via interaction with the Client License Manager 608 that the user's client system holds a license to run the application at the current time, checks the Client Stream Cache 611 and satisfies the page fault from that cache, if possible. If the page is not currently in the Client Stream Cache 611, the Client Cache Manager 606 makes a request to the Client/Server Network Interface 505, 609 to obtain that page from the Application File Pages stored on an Application Server 506.

[0170] The Client Prefetcher 606 tracks all page requests passed to the Client Cache Manager 606. Based on the pattern of those requests and on program locality or program history, the Client Prefetcher 606 asks the Client Cache Manager 606 to send additional requests to the Client/Server Network Interface 505, 609 to obtain other pages from the Application File Pages stored on the Application Server 506.

[0171] Files located on the Client Streaming File System 604 are typically identified by a particular prefix (like drive letter or pathname). However, some files whose names would normally imply that they reside locally are mapped to the Client Streaming File System 604, in order to lower the invention's impact on the user's local configuration. For instance, there are certain shared library files (dll's) that need to be installed on the local file system (c:\winnt\system32\foo.dll). It is undesirable to add that file on the user's system. The file name gets added to a "spoof database" which contains an entry saying that c:\winnt\system32\foo.dll is mapped to z:\word\winnt\system32\foo.dll where z: implies that it is the Client Streaming File System. The Client Spoofer 603 will then redirect all accesses to c:\winnt\system32\foo.dll to z:\word\winnt\system32\foo.dll. In this manner the client system gets the effect of the file being on the local machine whereas in reality the file is streamed from the server.

[0172] In a similar fashion the Client Spoofer 603 may also be used to handle mapping TCP interfaces to HTTP interfaces. There are certain client-server applications (like

ERP/CRM applications) that have a component running on a client and another component running on a database server, Web server etc. These components talk to each other through TCP connections. The client application will make TCP connections to the appropriate server (for this example, a database server) when the client piece of this application is being streamed on a user's machine.

[0173] The database server could be resident behind a firewall and the only way for the client and the server to communicate is through a protocol like HTTP that can pass through firewalls. To enable the client to communicate with the database server, the client's TCP requests need to be converted to HTTP and sent to the database server. Those requests can be converted back to TCP so that the database server can appropriately process the requests just before the requests reach the database server. The Client Spoofer's 603 responsibility in this case is to trap all TCP requests going to the database server and convert it into HTTP requests and take all HTTP requests coming from the database server and convert them into TCP packets. Note that the TCP to HTTP converters 505, 507 convert TCP traffic to HTTP and vice versa by embedding TCP packets within the HTTP protocol and by extracting the TCP packets from the HTTP traffic. This is called tunneling.

[0174] When the Client License Manager 608 is asked about a client's status with respect to holding a license for a particular application and the license is not already being held, the Client License Manager 608 contacts the License Server 106 via the Client/Server Network Interface 609 and asks that the client machine be given the license. The License Server 106 checks the Subscription 101 and License 102 Databases and, if the user has the right to hold the license at the current time, it sends back an Access Token, which represents the right to use the license. This Access Token is renewed by the client on a periodic basis.

[0175] The user sets up and updates his information in the Subscription 101 and License 102 Databases via interacting with the Subscription Server 105. Whenever a user changes his subscription information, the Subscription Server 105 signals the user's client system since the client's Known ASPs and Installed Apps information potentially needs updating. The client system also checks the Subscription 101 and License 102 Databases whenever the user logs into any of his client systems set up for Streaming Application Delivery and Execution. If the user's subscription list in the Subscription 101 and License 102 Databases list applications that have not been installed on the user's client system, the user is given the opportunity to choose to install those applications.

[0176] Whenever the user chooses to install an application, the Client License Manager 608 passes the request to the Client Application Installer 607 along with the name of the Stream App Install Block to be obtained from the Application Server 107. The Client Application Installer 607 opens and reads that file (which engages the Client Streaming File System) and updates the Client system appropriately, including setting up the spoof database, downloading certain needed non-application-specific files, modifying the registry file, and optionally providing a list of applications pages to be prefetched to warm up the Client Stream Cache 611 with respect to the application.

[0177] The Application Stream Builder creates the Stream App Install Block 405 used to set up a client system for Streaming Application Delivery and Execution and it also creates the set of Application File Pages 406 sent to satisfy client requests by the Application Server 107. The process that creates this information is offline and involves three components. The Application Install Monitor 403 watches a normal installation of the application and records various information including registry entries, required system configuration, file placement, and user options. The Application Profiler 407 watches a normal execution of the application and records referenced pages, which may be requested to pre-warm the client's cache on behalf of this application. The Application Stream Packager 404 takes information from the other two Builder components, plus some information it compiles with respect to the layout of the installed application and forms the App Install Block 405 and the set of Application File Pages 406.

[0178] Server fail-over and server quality of service problems are handled by the client via observation and information provided by the server components. An ASP's Subscription Server provides a list of License Servers associated with that ASP to the client, when the user initiates/modifies his account or when the client software explicitly requests a new list. A License Server provides a list of Application Servers associated with an application to the client, whenever it sends the client an Access Token for the application.

[0179] Should the client observe apparent non-response or slow response from an Application Server, it switches to another Application Server in its list for the application in question. If none of the Application Servers in its list respond adequately, the client requests a new set for the application from a License Server. The strategy is similar in the case in which the client observes apparent non-response or slow response from a License Server; the client switches to another License Server in its list for the ASP in question. If none of the License Servers in its list responds adequately, the client requests a new set of License Servers from the ASP.

[0180] Server load balancing is handled by the server components in cooperation with the client. A server monitor component tracks the overall health and responsiveness of all servers. When a server is composing one of the server lists mentioned in the previous paragraph, it selects a set that is alive and relatively more lightly used than others. Client cooperation is marked by the client using the server lists provided by the servers in the expected way, and not unilaterally doing something unexpected, like continuing to use a server which does not appear in the most recent list provided.

[0181] Security issues associated with the server client relationship are considered in the invention. To ensure that the communication between servers and clients is private and that the servers in question are authorized via appropriate certification, an SSL layer is used. To ensure that the clients are licensed to use a requested application, user credentials (username+password) are presented to a License Server, which validates the user and his licensing status with respect to the application in question and issues an Access Token, and that Access Token is in turn presented to an Application Server, which verifies that the Token's validity before delivering the requested page. Protecting the application in question from piracy on the client's system is discussed in another section, below.

CLIENT-SIDE PERFORMANCE OPTIMIZATION

[0182] This section focuses on client-specific portions of the invention. The invention may be applied to any operating system that provides a file system interface or block driver interface. A preferred embodiment of the invention is Windows 2000 compliant.

[0183] With respect to FIG. 6a, several different components of the client software are shown. Some components will typically run as part of the operating system kernel, and other portions will run in user mode.

[0184] The basis of the client side of the streamed application delivery and execution system is a mechanism for making applications appear as though they were installed on the client computer system without actually installing them.

[0185] Installed applications are stored in the file system of the client system as files organized in directories. In the state of the art, there are two types of file systems: local and network. Local file systems are stored entirely on media (disks) physically resident in the client machine. Network file systems are stored on a machine physically separate from the client, and all requests for data are satisfied by getting the data from the server. Network file systems are typically slower than local file systems. A traditional approach to use the better performance of a local file system is to install important applications on the local file system, thereby copying the entire application to the local disk. The disadvantages of this approach are numerous. Large applications may take a significant amount of time to download, especially across slower wide area networks. Upgrading applications is also more difficult, since each client machine must individually be upgraded.

[0186] The invention eliminates these two problems by providing a new type of file system: a streaming file system. The streaming file system allows applications to be run immediately by retrieving application file contents from the server as they are needed, not as the application is installed. This removes the download cost penalty of doing local installations of the application. The streaming file system also contains performance enhancements that make it superior to running applications directly from a network file system. The streaming file system caches file system contents on the local machine. File system accesses that hit in the cache are nearly as fast as those to a local file system. The streaming file system also has sophisticated information about application file access patterns. By using this knowledge, the streaming file system can request portions of application files from the server in advance of when they will actually be needed, thus further improving the performance of applications running on the application streaming file system.

[0187] In a preferred embodiment of the invention, the application streaming file system is implemented on the client using a file system driver and a helper application running in user mode. The file system driver receives all requests from the operating system for files belonging to the application streaming file system. The requests it handles are all of the standard file system requests that every file system must handle, including (but not limited to) opening and closing files, reading and writing files, renaming files, and deleting files. Each file has a unique identifier consisting of an application number, and a file number within that appli-

cation. In one embodiment of the invention, the application number is 128 bits and the file number is 32 bits, resulting in a unique file ID that is 160 bits long. The file system driver is responsible for converting path names (such as "z:\program files\foo.exe") into file IDs (this is described below). Once the file system driver has made this translation, it basically forwards the request to the user-mode program to handle.

[0188] The user-mode program is responsible for managing the cache of application file contents on the local file system and contacting the application streaming server for file contents that it cannot satisfy out of the local cache. For each file system request, such as read or open, the user-mode process will check to see if it has the requested information in the cache. If it does, it can copy the data from the cache and return it to the file system driver. If it does not, it contacts the application streaming server over the network and obtains the information it needs. To obtain the contents of the file, the user-mode process sends the file identifier for the file it is interested in reading along with an offset at which to read and the number of bytes to read. The application streaming server will send back the requested data.

[0189] The file system can be implemented using a fragmented functionality to facilitate development and debugging. All of the functionality of the user-mode component can be put into the file system driver itself without significantly changing the scope of the invention. Such an approach is believed to be preferred for a client running Windows 95 as the operating system.

[0190] Directories are specially formatted files. The file system driver reads these from the user mode process just like any other files with reads and writes. Along with a header containing information about the directory (such as how long it is), the directory contains one entry for each file that it contains. Each entry contains the name of the file and its file identifier. The file identifier is necessary so that the specified file can be opened, read, or written. Note that since directories are files, directories may recursively contain other directories. All files in an application streaming file system are eventual descendants of a special directory called the "root". The root directory is used as the starting point for parsing file names.

[0191] Given a name like "z:/foo/bar/baz", the file system driver must translate the path "z:/foo/bar/baz" into a file identifier that can be used to read the file from the application streaming service. First, the drive letter is stripped off, leaving "/foo/bar/baz". The root directory will be searched for the first part of the path, in this case "foo". If the file "foo" is found in the root directory, and the file "foo" is a directory, then "foo" will be searched for the next portion of the path, "bar". The file system driver achieves this by using the file id for "foo" (found by searching the root directory) to open the file and read its contents. The entries inside "foo" are then searched for "bar", and this process continues until the entire path is parsed, or an error occurs.

[0192] In the following examples and text, the root directory is local and private to the client. Each application that is installed will have its own special subdirectory in the root directory. This subdirectory will be the root of the application. Each application has its own root directory.

[0193] The invention's approach is much more efficient than other approaches like the standard NFS approach. In those cases, the client sends the entire path "/foo/bar/baz" to the server and the server returns the file id for that file. The next time there is a request for "/foo/bar/baz2" the entire path again needs to be sent. In the approach described here, once the request for "bar" was made, the file ids for all files within bar are sent back including the ids for "baz" and "baz2" and hence "baz2" will already be known to client. This reduces communication between the client and the server.

[0194] In addition, this structure also allows applications to be easily updated. If certain code segments need to be updated, then the code segment listing in the application root directory is simply changed and the new code segment subdirectory added. This results in the new and correct code segment subdirectory being read when it is referenced. For example if a file by the name of "/foo/bar/baz3" needs to be added, the root directory is simply changed to point to a new version of "foo" and that new version of "foo" points to a new version of "bar" which contains "baz3" in addition to the files it already contained. However the rest of the system is unchanged.

[0195] Client Features

[0196] Referring to FIGS. 6a and 6b, a key aspect of the preferred embodiment of the invention is that application code and data are cached in the client's persistent storage 616, 620. This caching provides better performance for the client, as accessing code and data in the client's persistent storage 620 is typically much faster than accessing that data across a wide area network. This caching also reduces the load on the server, since the client need not retrieve code or data from the application server that it already has in its local persistent storage.

[0197] In order to run an application, its code and data must be present in the client system's volatile storage 619. The client software maintains a cache of application code and data that normally reside in the client system's non-volatile memory 620. When the running application requires data that is not present in volatile storage 619, the client streaming software 604 is asked for the necessary code or data. The client software first checks its cache 611, 620 in nonvolatile storage for the requested code or data. If it is found there, the code or data are copied from the cache in nonvolatile storage 620 to volatile memory 619. If the requested code or data are not found in the nonvolatile cache 611, 620, the client streaming software 604 will acquire the code or data from the server system via the client's network interface 621, 622.

[0198] Application code and data may be compressed 623, 624 on the server to provide better client performance over slow networks. Network file systems typically do not compress the data they send, as they are optimized to operate over local area networks.

[0199] FIGS. 7a & 7b demonstrate two ways in which data may be compressed while in transit between the server and client. With either mechanism, the client may request multiple pieces of code and data from multiple files at once. FIG. 7A illustrates the server 701 compressing the concatenation of A, B, C, and D 703 and sending this to the client 702. FIG. 7B illustrates the server 706 separately compress-

ing A, B, C, and D 708 and sending the concatenation of these compressed regions to the client 707. In either case, the client 702, 707 will decompress the blocks to retrieve the original contents A, B, C, and D 704, 709 and these contents will be stored in the cache 705, 710.

[0200] The boxes marked "Compression" represent any method of making data more compact, including software algorithms and hardware. The boxes marked "Decompression" represent any method for expanding the compacted data, including software algorithms and hardware. The decompression algorithm used must correspond to the compression algorithm used.

[0201] The mechanism for streaming of application code and data may be a file system. Many network file systems exist. Some are used to provide access to applications, but such systems typically operate well over a local area network (LAN) but perform poorly over a wide area network (WAN). While this solution involves a file system driver as part of the client streaming software, it is more of an application delivery mechanism than an actual file system.

[0202] With respect to FIG. 8, application code and data are installed onto the file system 802, 805, 806, 807 of a client machine, but they are executed from the volatile storage (main memory). This approach to streamed application delivery involves installing a special application streaming file system 803, 804. To the client machine, the streaming file system 803, 804 appears to contain the installed application 801. The application streaming file system 803 will receive all requests for code or data that are part of the application 801. This file system 803 will satisfy requests for application code or data by retrieving it from its special cache stored in a native file system or by retrieving it directly from the streaming application server 802. Code or data retrieved from the server 802 will be placed in the cache in case it is used again.

[0203] Referring to FIG. 9, an alternative organization of the streaming client software is shown. The client software is divided into the kernel-mode streaming file system driver 905 and a user-mode client 902. Requests made to the streaming file system driver 905 are all directed to the user-mode client 902, which handles the streams from the application streaming server 903 and sends the results back to the driver 905. The advantage of this approach is that it is easier to develop and debug compared with the pure-kernel mode approach. The disadvantage is that the performance will be worse than that of a kernel-only approach.

[0204] As shown in FIGS. 10 and 11, the mechanism for streaming of application code and data may be a block driver 1004, 1106. This approach is an alternative to that represented by FIGS. 8 and 9.

[0205] With respect to FIG. 10, the application streaming software consists of a streaming block driver 1004. This block driver 1004 provides the abstraction of a physical disk to a native file system 1003 already installed on the client operating system 1002. The driver 1004 receives requests for physical block reads and writes, which it satisfies out of a cache on a standard file system 1003 that is backed by a physical disk drive 1006, 1007. Requests that cannot be satisfied by the cache go to the streaming application server 1005, as before.

[0206] Referring to FIG. 11, the application streaming software has been divided into a disk driver 1106 and a user mode client 1102. In a manner similar to that of FIG. 9, the disk driver 1106 sends all requests it gets to the user-mode client 1102, which satisfies them out of the cache 1107, 1108 or by going to the application streaming server 1103.

[0207] The persistent cache may be encrypted with a key not permanently stored on the client to prevent unauthorized use or duplication of application code or data. Traditional network file systems do not protect against the unauthorized use or duplication of file system data.

[0208] With respect to FIG. 12, unencrypted and encrypted client caches. A, B, C, and D 1201 representing blocks of application code and data in their natural form are shown. $E_k(X)$ represents the encryption of block X with key k 1202. Any encryption algorithm may be used. The key k is sent to the client upon application startup, and it is not stored in the application's persistent storage.

[0209] Client-initiated prefetching of application code and data helps to improve interactive application performance. Traditional network file systems have no prefetching or simple locality based prefetching.

[0210] Referring to FIG. 13, the application 1301 generates a sequence of code or data requests 1302 to the operating system(OS) 1303. The OS 1303 directs these 1304 to the client application streaming software 1305. The client software 1305 will fetch the code or data 1306 for any requests that do not hit in the cache from the server 1307, via the network. The client software 1305 inspects these requests and consults the contents of the cache 1309 as well as historic information about application fetching patterns 1308. It will use this information to request additional blocks of code and data that it expects will be needed soon. This mechanism is referred to as "pull prefetching."

[0211] Server-initiated prefetching of application code and data helps to improve interactive application performance. Traditional network file systems have no prefetching or simple locality based prefetching.

[0212] With respect to FIG. 14, the server-based prefetching is shown. As in FIG. 13, the client application streaming software 1405 makes requests for blocks 1407 from the application streaming server 1408. The server 1408 examines the patterns of requests made by this client and selectively returns to the client additional blocks 1406 that the client did not request but is likely to need soon. This mechanism is referred to as "push prefetching."

[0213] A client-to-client communication mechanism allows local application customization to travel from one client machine to another without involving server communication. Some operating systems have a mechanism for copying a user's configuration and setup to another machine. However, this mechanism typically doesn't work outside of a single organization's network, and usually will copy the entire environment, even if only the settings for a single application are desired.

[0214] Referring to FIG. 15, a client-to-client mechanism is demonstrated. When a user wishes to run an application on a second machine, but wishes to retain customizations made previously on the first, the client software will handle this by contacting the first machine to retrieve customized files and other customization data. Unmodified files will be retrieved as usual from the application streaming server.

[0215] Here, File 4 exists in three different versions. The server 1503 provides one version of this file 1506, client 11501 has a second version of this file 1504, and client 21502 has a third version 1505. Files may be modified differently for each client.

[0216] The clients may also contain files not present on the server or on other clients. File 51507 is one such file; it exists only on client 11501. File 61508 only exists on client 21502.

[0217] Local Customization

[0218] A local copy-on-write file system allows some applications to write configuration or initialization files where they want to without rewriting the application, and without disturbing the local customization of other clients. Installations of applications on file servers typically do not allow the installation directories of applications to be written, so additional reconfiguration or rewrites of applications are usually necessary to allow per-user customization of some settings.

[0219] With respect to FIG. 16, the cache 1602 with extensions for supporting local file customization is shown. Each block of data in the cache is marked as "clean" 1604 or "dirty" 1605. Pages marked as dirty have been customized by the client 1609, and cannot be removed from the cache 1602 without losing client customization. Pages marked as clean may be purged from the cache 1602, as they can be retrieved again from the server 1603. The index 1601 indicates which pages are clean and dirty. In FIG. 16, clean pages are white, and dirty pages are shaded. File 11606 contains only clean pages, and thus may be entirely evicted from the cache 1602. File 21607 contains only dirty pages, and cannot be removed at all from the cache 1602. File 31608 contains some clean and some dirty pages 1602. The clean pages of File 31608 may be removed from the cache 1602, while the dirty pages must remain.

[0220] Selective Write Protection

[0221] The client streaming software disallows modifications to certain application files. This provides several benefits, such as preventing virus infections and reducing the chance of accidental application corruption. Locally installed files are typically not protected in any way other than conventional backup. Application file servers may be protected against writing by client machines, but are not typically protected against viruses running on the server itself. Most client file systems allow files to be marked as read-only, but it is typically possible to change a file from read-only to read-write. The client application streaming software will not allow any data to be written to files that are marked as not modifiable. Attempts to mark the file as writable will not be successful.

[0222] Error Detection and Correction

[0223] The client streaming software maintains checksums of application code and data and can repair damaged or deleted files by retrieving another copy from the application streaming server. Traditional application delivery mechanisms do not make any provisions for detecting or correcting corrupted application installs. The user typically detects a corrupt application, and the only solution is to completely reinstall the application. Corrupt application files are detected by the invention automatically, and replacement code or data are invisibly retrieved by the client streaming software without user intervention.

[0224] When a block of code or data is requested by the client operating system, the client application streaming software will compute the checksum of the data block before it is returned to the operating system. If this checksum does not match that stored in the cache, the client will invalidate the cache entry and retrieve a fresh copy of the page from the server.

[0225] File Identifiers

[0226] Applications may be patched or upgraded via a change in the root directory for that application. Application files that are not affected by the patch or upgrade need not be downloaded again. Most existing file systems do not cache files locally.

[0227] Each file has a unique identifier (number). Files that are changed or added in the upgrade are given new identifiers never before used for this application. Files that are unchanged keep the same number. Directories whose contents change are also considered changes. If any file changes, this will cause its parent to change, all the way up to the root directory.

[0228] Upgrade Mechanism

[0229] When the client is informed of an upgrade, it is told of the new root directory. It uses this new root directory to search for files in the application. When retrieving an old file that hasn't changed, it will find the old file identifier, which can be used for the existing files in the cache. In this way, files that do not change can be reused from the cache without downloading them again. For a file that has changed, when the file name is parsed, the client will find a new file number. Because this file number did not exist before the upgrade, the client will not have this file in the cache, and will stream the new file contents when the file is freshly accessed. This way it always gets the newest version of files that change.

[0230] The client application streaming software can be notified of application upgrades by the streaming application server. These upgrades can be marked as mandatory, in which case the client software will force the application to be upgraded.

[0231] The client will contact the application streaming server when it starts the application. At this time, the streaming application server can inform the client of any upgrades. If the upgrade is mandatory, the client will be informed, and it will automatically begin using the upgraded application by using the new root directory.

[0232] Multicast Technique

[0233] A broadcast or multicast medium may be used to efficiently distribute applications from one application streaming server to multiple application streaming clients. Traditional networked application delivery mechanisms usually involve installing application code and data on a central server and having client machines run the application from that server. The multicast mechanism allows a single server to broadcast or multicast the contents of an application to many machines simultaneously. The client machines will receive the application via the broadcast and save it in their local disk cache. The entire application can be distributed to a large number of client machines from a single server very efficiently.

[0234] The multicast network is any communication mechanism that has broadcast or multicast capability. Such media include television and radio broadcasts and IP mul-

ticasting on the Internet. Each client that is interested in a particular application may listen to the multicast media for code and data for that application. The code and data are stored in the cache for later use when the application is run.

[0235] These client techniques can be used to distribute data that changes rarely. Application delivery is the most appealing use for these techniques, but they could easily be adopted to distribute other types of slowly changing code and data, such as static databases.

LOAD BALANCING AND FAULT TOLERANCE FOR STREAMED APPLICATIONS

[0236] This section focuses on load balancing (and thereby scalability) and hardware fail over. Throughout this discussion reference should be made to FIG. 17. Load balancing and fault tolerance are addressed in the invention by using a smart client and smart server combination. A preferred embodiment of the invention that implements these features includes three types of servers (described below): app servers; SLM servers; and an ASP Web server. These are organized as follows:

[0237] 1: ASP Web server 1703—This is the Web server that the user goes to for subscribing to applications, creating accounts etc. Compared to the other two types of servers it is characterized by: lowest traffic, fewest number of them, & least likely to go down.

[0238] 2: SLM Servers 1707—subscription license manager servers—These keep track of which user has subscribed to what applications under what license etc. Compared to the other two types of servers it is characterized by: medium traffic, manageable number, and less likely to go down.

[0239] 3: App Servers 1710—These are the servers to which the users go to for application pages. Compared to the other two types of servers it is characterized by: highest traffic, most number of them, most likely to go down either due to hardware failure or application re-configuration.

[0240] Server Lists

[0241] Clients 1704 subscribe and unsubscribe to applications via the ASP Web server 1703. At that point, instead of getting a primary and a secondary server that can perform the job, the ASP Web server 1703 gives them a non-prioritized list of a large number of SLM servers 1706 that can do the job. When the application starts to run, each client contacts the SLM servers 1707, 1708, 1709 and receive its application server list 1705 that can serve the application in question and also receive the access tokens that can be used to validate themselves with the application servers 1710-1715. All access tokens have an expiration time after which they need to be renewed.

[0242] Server Selection

[0243] Having gotten a server list for each type of server 1705, 1706, the client 1704 will decide which specific server to send its request to. In a basic implementation, a server is picked randomly from the list, which will distribute the client's load on the servers very close to evenly. An alternative preferred implementation will do as follows:

[0244] a) Clients will initially pick servers from the list randomly, but they will also keep track of the overall response time they get from each request; and

[0245] b) As each client learns about response times for each server, it can be more intelligent (rather than random) and pick the most responsive server. It is believed that the client is better suited at deciding which server is most responsive because it can keep track of the round trip response time.

[0246] Client-side Hardware Fail Over

[0247] The server selection logic provides hardware failover in the following manner:

[0248] a) If a server does not respond, i.e., times out, the client 1704 will pick another server from its list 1705, 1706 and re-send the request. Since all the servers in the client's server list 1705, 1706 are capable of processing the client's request, there are no questions of incompatibility.

[0249] b) If a SAS client 1704 gets a second time out, i.e., two servers are down, it re-sends the request to multiple servers from its list 1705, 1706 in parallel. This approach staggers the timeouts and reduces the overall delay in processing a request.

[0250] c) In case of a massive hardware failure, all servers in the client's list 1705, 1706 may be down. At this point, the client will use the interfaces to refresh its server list. This is where the three tiers of servers become important:

[0251] 1) If the client 1704 want to refresh its App server list 1705, it will contact an SLM server 1707, 1709 in its list of SLM servers 1706. Again, the same random (SLM) server selection order is utilized here. Most of the time, this request will be successful and the client will get an updated list of app servers.

[0252] 2) If for some reason all of the SLM servers 1707, 1709 in the client's list 1706 are down, it will contact the ASP Web server 1703 to refresh its SLM server list 1706.

[0253] This 3-tiered approach significantly reduces the impact of a single point of failure—the ASP Web server 1703, effectively making it a fail over of a fail over.

[0254] Server Load Balancing

[0255] In a preferred embodiment of the invention, a server side monitor 1702 keeps track of the overall health and response times for each server request. The Monitor performs this task for all Application and SLM servers. It posts prioritized lists of SLM servers and app servers 1701 that can serve each of the apps in a database shared by the monitor 1702 and all servers. The monitor's algorithm for prioritizing server lists is dominated by the server's response time for each client request. If any servers fail, the monitor 1702 informs the ASP 1703 and removes it from the server list 1701. Note that the server lists 1705, 1706 that the client 1704 maintains are subsets of lists the monitor 1702 maintains in a shared database 1701.

[0256] Since all servers can access the shared database 1701, they know how to 'cut' a list of servers to a client. For example, the client starts to run an SAS application or it wants to refresh its app server list: It will contact an SLM

server and the S L M server will access the database 1701 and cut a list of servers that are most responsive (from the server's prospective).

[0257] In this scheme, the server monitor 1702 is keeping track of what it can track the best: how effectively servers are processing client requests (server's response time). It does not track the network propagation delays etc. that can significantly contribute to a client's observed response time.

[0258] ASP Managing Hardware Failovers

[0259] The foregoing approaches provide an opportunity for ASPs to better manage massive scale failures. Specifically, when an ASP 1703 realizes that massive numbers of servers are down, it can allocate additional resource on a temporary basis. The ASP 1703 can update the central database 1701 such that clients will receive only the list that the ASP 1703 knows to be up and running. This includes any temporary resources added to aid the situation. A particular advantage of this approach is that ASP 1703 doesn't need special actions, e.g., emails or phone support, to route clients over to these temporary resources; the transition happens automatically.

[0260] Handling Client Crashes and Client Evictions

[0261] To prevent the same user from running the same application from multiple machines, the SLM servers 1707, 1708, 1709 track what access tokens have been handed to what users. The SAS file system tracks the beginning and end of applications. The user's SAS client software asks for an access token from the SLM servers 1707, 1708, 1709 at the beginning of an application if it already does not have one and it releases the access token when the application ends. The SLM server makes sure that at a given point only one access token has been given to a particular user. In this manner, the user can run the application from multiple machines, but only from one at a particular time. However, if the user's machine crashes before the access token has been relinquished or if for some reason the ASP 1703 wants to evict a user, the access token granted to the user must be made invalid. To perform this, the SLM server gets the list of application servers 1705 that have been sent to the client 1704 for serving the application and sends a message to those application servers 1710, 1711, 1713, 1714 to stop serving that particular access token. This list is always maintained in the database so that every SLM server can find out what list is held by the user's machine. The application servers before servicing any access token must check with this list to ensure that the access token has not become invalid. Once the access token expires, it can be removed from this list.

SERVER-SIDE PERFORMANCE OPTIMIZATION

[0262] This section describes approaches that can be taken to reduce client-side latency (the time between when an application page is needed and when it is obtained) and improve Application Server scalability (a measure of the number of servers required to support a given population of clients). The former directly affects the perceived performance of an application by an end user (for application features that are not present in the user's cache), while the latter directly affects the cost of providing application streaming services to a large number of users.

[0263] Application Server Operation

[0264] The basic purpose of the Application Server is to return Application File Pages over the network as requested by a client. The Application Server holds a group of Stream Application Sets from which it obtains the Application File Pages that match a client request. The Application Server is analogous to a typical network file system (which also returns file data), except it is optimized for delivery of Application file data, i.e., code or data that belong directly to the application, produced by the software provider, as opposed to general user file data (document files and other content produced by the users themselves). The primary differences between the Application Server and a typical network file system are:

[0265] 1. The restriction to handle only Application file data allows the Application Server to only service read requests, with writes being disallowed or handled on the client itself in a copy-on-write manner;

[0266] 2. Access checks occur at the application level, that is a client is given all-or-none access to files for a given software application;

[0267] 3. The Application Server is designed to operate across the Internet, as opposed to typical network file systems, which are optimized to run over LANs. This brings up additional requirements of handling server failures, maximizing network bandwidth and minimizing latency, and handling security; and

[0268] 4. The Application Server is application-aware, unlike typical network file systems, which treat all software application files the same as all other files. This allows the Application Server to use and collect per-application access profile information along with other statistics.

[0269] To service a client request, the Application Server software component keeps master copies of the full Application Stream Sets on locally accessible persistent storage. In main memory, the Application Server maintains a cache of commonly accessed Application File Pages. The primary steps taken by the Application Server to service a client request are:

[0270] 1. Receive and decode the client request;

[0271] 2. Validate the client's privilege to access the requested data, e.g., by means of a Kerberos-style ticket issued by a trusted security service;

[0272] 3. Look up the requested data in the main memory cache, and failing that, obtain it from the master copy on disk while placing it in the cache; and

[0273] 4. Return the File Pages to the client over the network.

[0274] The techniques used to reduce latency and improve server scalability (the main performance considerations) are described below.

[0275] Server Optimization Features

[0276] Read-Only File System for Application Files—Because virtually all application files (code and data) are never written to by users, virtually the entire population of users have identical copies of the application files. Thus a

system intending to deliver the application files can distribute a single, fixed image across all servers. The read-only file system presented by the Application Server represents this sharing, and eliminates the complexities of replication management, e.g., coherency, that occur with traditional network file systems. This simplification enables the Application Servers to respond to requests more quickly, enables potential caching at intervening nodes or sharing of caches across clients in a peer-to-peer fashion, and facilitates fail over, since with the read-only file system the Application File Pages as identified by the client (by a set of unique numbers) will always globally refer to the same content in all cases.

[0277] Per-page Compression—Overall latency observed by the client can be reduced under low-bandwidth conditions by compressing each Application File Page before sending it. Referring to FIG. 18, the benefits of the use of compression in the streaming of Application File Pages, is illustrated. The client 1801 and server 1802 timelines are shown for a typical transfer of data versus the same data sent in a compressed form. The client requests the data from the server 1803. The server processes the request 1804 and begins sending the requested data. The timelines then diverge due to the ability to stream the compressed data 1805 faster than the uncompressed data 1806.

[0278] With respect to FIG. 19, the invention's pre-compression of Application File Pages process is shown. The Builder generates the stream application sets 1901, 1902 which are then pre-compressed by the Stream Application Set Post-Processor 1903. The Stream Application Set Post-Processor 1903 stores the compressed application sets in the persistent storage device 1904. Any client requests for data are serviced by the Application Server which sends the pre-compressed data to the requesting client 1905. The reduction in size of the data transmitted over the network reduces the time to arrival (though at the cost of some processing time on the client to decompress the data). When the bandwidth is low relative to processing power, e.g., 256 kbps with a Pentium-III-600, this can reduce latency significantly.

[0279] Page-set Compression—When pages are relatively small, matching the typical virtual memory page size of 4 kB, adaptive compression algorithms cannot deliver the same compression ratios that they can for larger blocks of data, e.g., 32 kB or larger. Referring to FIG. 20, when a client 2001 requests multiple Application File Pages at one time 2002, the Application Server 2006 can concatenate all the requested pages and compress the entire set at once 2004, thereby further reducing the latency the client will experience due to the improved compression ratio. If the pages have already been compressed 2003, then the request is fulfilled from the cache 2007 where the compressed pages are stored. The server 2006 responds to the client's request through the transfer of the compressed pages 2005.

[0280] Post-processing of Stream Application Sets—The Application Server may want to perform some post processing of the raw Stream Application Sets in order to reduce its runtime-processing load, thereby improving its performance. One example is to pre-compress all Application File Pages contained in the Stream Application Sets, saving a great deal of otherwise repetitive processing time. Another possibility is to rearrange the format to suit the hardware and operating system features, or to reorder the pages to take advantage of access locality.

[0281] Static and Dynamic Profiling—With respect to FIG. 21, since the same application code is executed in conjunction with a particular Stream Application Set 2103 each time, there will be a high degree of temporal locality of referenced Application File Pages, e.g., when a certain feature is invoked, most if not all the same code and data is referenced each time to perform the operation. These access patterns can be collected into profiles 2108, which can be shipped to the client 2106 to guide its prefetching (or to guide server-based 2105 prefetching), and they can be used to pre-package groups of Application File Pages 2103, 2104 together and compress them offline as part of a post-processing step 2101, 2102, 2103. The benefit of the latter is that a high compression ratio can be obtained to reduce client latency without the cost of runtime server processing load (though only limited groups of Application File Pages will be available, so requests which don't match the profile would get a superset of their request in terms of the pre-compressed groups of Application File Pages that are available).

[0282] Fast Server-Side Client Privilege Checks—Referring to FIG. 22, having to track individual user's credentials, i.e., which Applications they have privileges to access, can limit server scalability since ultimately the per-user data must be backed by a database, which can add latency to servicing of user requests and can become a central bottleneck. Instead, a separate License Server 2205 is used to offload per-user operations to grant privileges to access application data, and thereby allow the two types of servers 2205, 2210 to scale independently. The License Server 2205 provides the client an Access Token (similar to a Kerberos ticket) that contains information about what application it represents rights for along with an expiration time. This simplifies the operations required by the Application Server 2210 to validate a client's privileges 2212. The Application Server 2210 needs only to decrypt the Access Token (or a digest of it) via a secret key shared 2209 with the License Server 2205 (thus verifying the Token is valid), then checking the validity of its contents, e.g., application identifier, and testing the expiration time. Clients 2212 presenting Tokens for which all checks pass are granted access. The Application Server 2210 needs not track anything about individual users or their identities, thus not requiring any database operations. To reduce the cost of privilege checks further, the Application Server 2210 can keep a list of recently used Access Tokens for which the checks passed, and if a client passes in a matching Access Token, the server need only check the expiration time, with no further decryption processing required.

[0283] Connection Management—Before data is ever transferred from a client to a server, the network connection itself takes up one and a half network round trips. This latency can adversely impact client performance if it occurs for every client request. To avoid this, clients can use a protocol such as HTTP 1.1, which uses persistent connections, i.e., connections stay open for multiple requests, reducing the effective connection overhead. Since the client-side file system has no knowledge of the request patterns, it will simply keep the connection open as long as possible. However, because traffic from clients may be bursty, the Application Server may have more open connections than the operating system can support, many of them being temporarily idle. To manage this, the Application Server can aggressively close connections that have been idle for a

period of time, thereby achieving a compromise between the client's latency needs and the Application Server's resource constraints. Traditional network file systems do not manage connections in this manner, as LAN latencies are not high enough to be of concern.

[0284] Application Server Memory Usage/Load Balancing—File servers are heavily dependent on main memory for fast access to file data (orders of magnitude faster than disk accesses). Traditional file servers manage their main memory as cache of file blocks, keeping the most commonly accessed ones. With the Application Server, the problem of managing main memory efficiently becomes more complicated due to there being multiple servers providing a shared set of applications. In this case, if each server managed its memory independently, and was symmetric with the others, then each server would only keep those file blocks most common to all clients, across all applications. This would cause the most common file blocks to be in the main memory of each and every Application server, and since each server would have roughly the same contents in memory, adding more servers won't improve scalability by much, since not much more data will be present in memory for fast access. For example, if there are application A (accessed 50% of the time), application B (accessed 40% of the time), and application C (accessed 10% of the time), and application A and B together consume more memory cache than a single Application Server has, and there are ten Application Servers, then none of the Application Servers will have many blocks from C in memory, penalizing that application, and doubling the number of servers will improve C's performance only minimally. This can be improved upon by making the Application Servers asymmetric, in that a central mechanism, e.g., system administrator, assigns individual Application Servers different Application Stream Sets to provide, in accordance with popularity of the various applications. Thus, in the above example, of the ten servers, five can be dedicated to provide A, four to B, and one to C, (any extra memory available for any application) making a much more effective use of the entire memory of the system to satisfy the actual needs of clients. This can be taken a step further by dynamically (and automatically) changing the assignments of the servers to match client accesses over time, as groups of users come and go during different time periods and as applications are added and removed from the system. This can be accomplished by having servers summarize their access patterns, send them to a central control server, which then can reassign servers as appropriate.

CONVERSION OF CONVENTIONAL APPLICATIONS TO ENABLE STREAMED DELIVERY AND EXECUTION

[0285] The Streamed Application Set Builder is a software program. It is used to convert locally installable applications into a data set suitable for streaming over a network. The streaming-enabled data set is called the Streamed Application Set (SAS). This section describes the procedure used to convert locally installable applications into the SAS.

[0286] The application conversion procedure into the SAS consists of several phases. In the first phase, the Builder program monitors the installation process of a local installation of the desired application for conversion. The Builder monitors any changes to the system and records those

changes in an intermediate data structure. After the application is installed locally, the Builder enters the second phase of the conversion. In the second phase, the Builder program invokes the installed application executable and obtains sequences of frequently accessed file blocks of this application. Both the Builder program and the client software use the sequence data to optimize the performance of the streaming process. Once the sequencing information is obtained, the Builder enters the final phase of the conversion. In this phase, the Builder gathers all data obtained from the first two phases and processes the data into the Streamed Application Set.

[0287] Detailed descriptions of the three phases of the Builder conversion process are described in the following sections. The three phases consist of installation monitoring (IM), application profiling (AP), and SAS packaging (SP). In most cases, the conversion process is general and applicable to all types of systems. In places where the conversion is OS dependent, the discussion is focused on the Microsoft Windows environment. Issues on conversion procedure for other OS environments are described in later sections.

[0288] Installation Monitoring (IM)

[0289] In the first phase of the conversion process, the Builder Installation Monitor (IM) component invokes the application installation program that installs the application locally. The IM observes all changes to the local computer during the installation. The changes may involve one or more of the following: changes to system or environment variables; and modifications, addition, or deletion of one or more files. Initial system variables, environment variables, and files are accounted for by the IM before the installation begins to give a more accurate picture of any changes that are observed. The IM records all changes to the variables and files in a data structure to be sent to the Builder's Streamed Application Packaging component. In the following paragraphs, detailed description of the Installation Monitor is described for Microsoft Windows environment.

[0290] In Microsoft Windows system, the Installation Monitor (IM) component consists of a kernel-mode driver subcomponent and a user-mode subcomponent. The kernel-mode driver is hooked into the system registry and file system function interface calls. The hook into the registry function calls allows the IM to monitor system variable changes. The hook into the file system function calls enables the IM to observe file changes.

[0291] Installation Monitor Kernel-Mode subcomponent (IM-KM)

[0292] With respect to FIG. 23, the IM-KM subcomponent monitors two classes of information during an application installation: system registry modifications and file modifications. Different techniques are used for each of these classes.

[0293] To monitor system registry modifications 2314, the IM-KM component replaces all kernel-mode API calls in the System Service Table that write to the system registry with new functions defined in the IM-KM subcomponent. When an installation program calls one of the API functions to write to the registry 2315, the IM-KM function is called instead, which logs the modification data 2317 (including registry key path, value name and value data) and then forwards the call to the actual operating system defined

function 2318. The modification data is made available to the IM-UM subcomponent through a mechanism described below.

[0294] To monitor file modifications, a filter driver is attached to the file system's driver stack. Each time an installation program modifies a file on the system, a function is called in the IM-KM subcomponent, which logs the modification data (including file path and name) and makes it available to the IM-UM using a mechanism described below.

[0295] The mechanisms used for monitoring registry modifications and file modifications will capture modifications made by any of the processes currently active on the computer system. While the installation program is running, other processes that, for example, operate the desktop and service network connections may be running and may also modify files or registry data during the installation. This data must be removed from the modification data to avoid inclusion of modifications that are not part of the application installation. The IM-KM uses process monitoring to perform this filtering.

[0296] To do process monitoring, the IM-KM installs a process notification callback function that is called each time a process is created or destroyed by the operating system. Using this callback function, the operating system sends the created process ID as well as the process ID of the creator (or parent) process. The IM-KM uses this information, along with the process ID of the IM-UM, to create a list of all of the processes created during the application installation. The IM-KM uses the following algorithm to create this list:

[0297] 1. Before the installation program is launched by the IM-UM, the IM-UM passes its own process ID to the IM-KM. Since the IM-UM is launching the installation application, the IM-UM will be the ancestor (parent, grandparent, etc.) of any process (with one exception—the Installer Service described below) that modifies files or registry data as part of the application installation.

[0298] 2. When the installation is launched and begins the creating processes, the IM-KM process monitoring logic is notified by the operating system via the process notification callback function.

[0299] 3. If the creator (parent) process ID sent to the process notification callback function is already in the process list, the new process is included in the list.

[0300] When an application on the system modifies either the registry or files, and the IM-KM monitoring logic captures the modification data, but before making it available to the IM-UM, it first checks to see if the process that modified the registry or file is part of the process list. It is only made available to the IM-UM if it is in the process list.

[0301] It is possible that a process that is not a process ancestor of the IM-UM will make changes to the system as a proxy for the installation application. Using interprocess communication, an installation program may request that an Installer Service make changes to the machine. In order for the IM-KM to capture changes made by the Installer Service, the process monitoring logic includes a simple rule that also includes any registry or file changes that have been

made by a process with the same name as the Installer Service process. On Windows 2000, for example, the Installer Service is called "msi.exe".

[0302] Installation Monitor User-Mode subcomponent (IM-UM)

[0303] The IM kernel-mode (IM-KM) driver subcomponent is controlled by the user-mode subcomponent (IM-UM). The IM-KM sends messages to the IM-UM to start 2305 and stop 2309 the monitoring process via standard I/O control messages known as IOCTLs. The message that starts the IM-KM also passes in the process ID of the IM-UM to facilitate process monitoring described in the IM-KM description.

[0304] When the installation program 2306 modifies the computer system, the IM-KM signals a named kernel event. The IM-UM listens for these events during the installation. When one of these events is signaled, the IM-KM calls the IM-KM using an IOCTL message. In response, the IM-KM packages data describing the modification and sends it to the IM-UM 2318.

[0305] The IM-UM sorts this data and removes duplicates. Also, it parameterizes all local-system-specific registry keys, value names, and values. For example, an application will often store paths in the registry that allow it to find certain files at run-time. These path specifications must be replaced with parameters that can be recognized by the client installation software.

[0306] A user interface is provided for the IM-UM that allows an operator of the Builder to browse through the changes made to the machine and to edit the modification data before the data is packaged into an SAS.

[0307] Once the installation of an application is completed 2308, the IM-UM forwards data structures representing the file and registry modifications to the Streamed Application Packager 2312.

[0308] Monitoring Application Configuration

[0309] Using the techniques described above for monitoring file modifications and monitoring registry modifications, the builder can also monitor a running application that is being configured for a particular working environment. The data acquired by the IM-UM can be used to duplicate the same configuration on multiple machines, making it unnecessary for each user to configure his/her own application installation.

[0310] An example of this is a client server application for which the client will be streamed to the client computer system. Common configuration modifications can be captured by the IM and packed into the SAS. When the application is streamed to the client machine, it is already configured to attach to the server and begin operation.

[0311] Application Profiling (AP)

[0312] Referring to FIG. 24, in the second phase of the conversion process, the Builder's Application Profiler (AP) component invokes the application executable program that is installed during the first phase of the conversion process. Given a particular user input, the executable program file blocks are accessed in a particular sequence. The purpose of the AP is to capture the sequence data associated with some user inputs. This data is useful in several ways.

[0313] First of all, frequently used file blocks can be streamed to the client machine before other less used file blocks. A frequently used file block is cached locally on the client cache before the user starts using the streamed application for the first time. This has the effect of making the streamed application as responsive to the user as the locally installed application by hiding any long network latency and bandwidth problems.

[0314] Secondly, the frequently accessed files can be reordered in the directory to allow faster lookup of the file information. This optimization is useful for directories with large number of files. When the client machine looks up a frequently used file in a directory, it finds this file early in the directory search. In an application run with many directory queries, the performance gain is significant.

[0315] Finally, the association of a set of file blocks with a particular user input allows the client machine to request minimum amount of data needed to respond to that particular user command. The profile data association with a user command is sent from the server to the client machine in the AppInstallBlock during the 'preparation' of the client machine for streaming. When the user on a client machine invokes a particular command, the codes corresponding to this command are prefetched from the server.

[0316] The Application Profiler (AP) is not as tied to the system as the Installation Monitor (IM) but there are still some OS dependent issues. In the Windows system, the AP still has two subcomponents: kernel-mode (AP-KM) sub-component and the user-mode (AP-UM) sub-component. The AP-UM invokes the converting application executable. Then AP-UM starts the AP-KM 2403, 2413 to track the sequences of file block accesses by the application 2414. Finally when the application exits after the pre-specified amount of sequence data is gathered, the AP-UM retrieves the data from AP-KM 2406, 2417 and forwards the data to the Streamed Application Packager 2411.

[0317] Streamed Application Set Packaging (SP)

[0318] With respect to FIG. 25, in the final phase of the conversion process, the Builder's Streamed Application Set Packager (SP) component processes the data structure from IM and AP to create a data set suitable for streaming over the network. This converted data set is called the Streamed Application Set 2520 and is suitable for uploading to the Streamed Application Servers for subsequent downloading by the stream client. FIG. 23 shows the control flow of the SP module.

[0319] Each file included in a Streamed Application Set 2520 is assigned a file number that identifies it within the SAS.

[0320] The Streamed Application Set 2520 consists of the three sets of data from the Streamed Application Server's perspective. The three types of data are the Concatenation Application File (CAF) 2519, 2515, the Size Offset File Table (SOFT) 2518, 2514, 2507, and the Root Versioning Table (RVT) 2518, 2514.

[0321] The CAF 2519, 2515 consists of all the files and directories needed to stream to the client. The CAF can be further divided into two subsets: initialization data set and the runtime data set.

[0322] The initialization data set is the first set of data to be streamed from the server to the client. This data set contains the information captured by IM and AP needed by the client to prepare the client machine for streaming this particular application. This initialization data set is also called the AppInstallBlock (AIB) 2516, 2512. In addition to the data captured by the IM and AP modules, the SP is also responsible for merging any new dynamic profile data gathered from the client and the server. This data is merged into the existing AppInstallBlock to optimize subsequent streaming of the application 2506. With the list of files obtained by the IM during application installation, the SP module separates the list of files into regular streamed files and the spoof files. The spoof files consists of those files not installed into standard application directory. This includes files installed into system directories and user specific directories. The detailed format description of the AppInstallBlock is described later.

[0323] The second part of the CAF consists of the runtime data set. This is the rest of the data that is streamed to the client once the client machine is initialized for this particular application. The runtime data consists of all the regular application files and the directories containing information about those application files. Detailed format description of the runtime data in the CAF section is described below. The SP appends every file recorded by IM into the CAF and generates all directories. Each directory contains list of file name, file number, and the metadata associated with the files in that particular directory 2517, 2513.

[0324] The SP is also responsible for generating the SOFT file 2518, 2514, 2507. This is a table used to index into the CAF for determining the start and the end of a file. The server uses this information to quickly access the proper file within the directory for serving the proper file blocks to the client.

[0325] Finally, the SP creates the RVT file 2518, 2514. The Root Versioning Table contains a list of root file number and version number. This information is used to track minor application patches and upgrades. Each entry in the RVT corresponds to one patch level of the application with a corresponding new root directory. The SP generates new parent directories when any single file in that subdirectory tree is changed from the patched upgrade. The RVT is uploaded to the server and requested by the client at appropriate time for the most updated version of the application by a simple comparison of the client's Streamed Application root file number with the RVT table located on the server once the client is granted access authorization to retrieve the data.

[0326] With respect to FIGS. 26a and 26b, the internal representation of a simple SAS before and after a new file is added to a new version of an application is shown. The original CAF 2601 has the new files 2607 appended to it 2604 by the SP. The SOFT 2602 is correspondingly updated 2605 with the appropriate table entries 2608 to index the new files 2607 the CAF 2604. Finally, the RVT 2603 is updated 2606 to reflect the new version 2609.

[0327] Data Flow Description

[0328] The following list describes the data that is passed from one component to another. The numbers corresponds to the numbering in the Data Flow diagram of FIG. 27.

[0329] Install Monitor

[0330] 1. The full pathname of the installer program is queried from the user by the Builder program and is sent to the Install Monitor.

[0331] 2. The Install Monitor (IM) user-mode sends a read request to the OS to spawn a new process for installing the application on the local machine.

[0332] 3. The OS loads the application installer program into memory and runs the application installer program. OS returns the process ID to the IM.

[0333] 4. The application program is started by the IM-UM.

[0334] 5. The application installer program sends read request to the OS to read the content of the CD.

[0335] 6. The CD media data files are read from the CD.

[0336] 7. The files are written to the appropriate location on the local hard-drive.

[0337] 8. IM kernel-mode captures all file read/write requests and all registry read/write requests by the application installer program.

[0338] 9. IM user-mode program starts the IM kernel-mode program and sends the request to start capturing all relevant file and registry data.

[0339] 10. IM kernel-mode program sends the list of all file modifications, additions, and deletions; and all registry modifications, additions, and deletions to the IM user-mode program.

[0340] 11. IM informs the SAS Builder UI that the installation monitoring has completed and displays the file and registry data in a graphical user interface.

[0341] Application Profiler

[0342] 12. Builder UI invokes Application Profiling (AP) user-mode program by querying the user for the list of application executable names to be profiled. The AP user-mode also queries the user for division of file blocks into sections corresponding to the commands invoked by the user of the application being profiled.

[0343] 13. Application Profiler user-mode invokes each application executable in succession by spawning each program in a new process. The OS loads the application executable into memory, runs the application executable, and returns the process ID to the Application Profiler.

[0344] 14. During execution, the OS on behalf of the application, sends the request to the hard-drive controller to read the appropriate file blocks into memory as needed by the application.

[0345] 15. The hard-drive controller returns all file blocks requested by the OS.

[0346] 16. Every file access to load the application file blocks into memory is monitored by the Application Profiler (AP) kernel-mode program.

[0347] 17. The AP user-mode program informs the AP kernel-mode program to start monitoring relevant file accesses.

[0348] 18. Application Profiler kernel-mode returns the file access sequence and frequency information to the user-mode program.

[0349] 19. Application Profiler returns the processed profile information. This has two sections. The first section is used to identify the frequency of files accessed. The second section is used to list the file blocks for prefetch to the client. The file blocks can be further categorized into subsections according to the commands invoked by the user of the application.

[0350] SAS Packager

[0351] 20. The Streamed Application Packager receives files and registry changes from the Builder UI. It also receives the file access frequency and a list of file blocks from the Profiler. File numbers are assigned to each file.

[0352] 21. The Streamed Application Packager reads all the file data from the hard-drive that are copied there by the application installer.

[0353] 22. The Streamed Application Packager also reads the previous version of Streamed Application Set for support of minor patch upgrades.

[0354] 23. Finally, the new Streamed Application Set data is stored back to non-volatile storage.

[0355] 24. For new profile data gathered after the SAS has been created, the packager is invoked to update the AppInstallBlock in the SAS with the new profile information.

[0356] Mapping of Data Flow to Streamed Application Set (SAS)

[0357] Step 7: Data gathered from this step consist of the registry and file modification, addition, and deletion. The data are mapped to the AppInstallBlock's File Section, Add Registry Section, and Remove Registry Section.

[0358] Step 8 & 19: File data are copied to the local hard-drive then concatenated into part of the CAF contents. Part of the data is identified as spoof or copied files and the file names and/or contents are added to the AppInstallBlock.

[0359] Step 15 & 21: Part of the data gathered by the Profiler or gathered dynamically by the client is used in the AppInstallBlock as a prefetch hint to the client. Another part of the data is used to generate a more efficient SAS Directory content by ordering the files according the usage frequency.

[0360] Step 20: If the installation program was an upgrade, SAS Packager needs previous version of the Streamed Application Set data. Appropriate

data from the previous version are combined with the new data to form the new Streamed Application Set.

[0361] Format of Streamed Application Set

[0362] Referring to FIG. 28, the format of the Streamed Application Set consists of three sections: Root Version Table (RVT) 2802, Size Offset File Table (SOFT) 2803, and Concatenation Application File (CAF) 2801. The RVT section 2802 lists all versions of the root file numbers available in a Streamed Application Set. The SOFT 2803 section consists of the pointers into the CAF 2801 section for every file in the CAF 2801. The CAF section 2801 contains the concatenation of all the files associated with the streamed application. The CAF section 2801 is made up of regular application files, SAS directory files 2805, ApplInstallBlock 2804, and icon files. See below for detailed information on the content of the SAS file.

[0363] OS Dependent Format

[0364] The format of the Streamed Application Set is designed to be as portable as possible across all OS platforms. At the highest level, the format of CAF, SOFT, and RVT that make up the format of the Streamed Application Set are completely portable across any OS platforms. One piece of data structure that is OS dependent is located in the initialization data set called ApplInstallBlock in the CAF. This data is dependent on the type of OS due to the differences in low-level system differences among different OS. For example, Microsoft Windows contains system environment variables called the Registry. The Registry has a particular tree format not found in other operating systems like UNIX or MacOS.

[0365] Another OS dependent piece of data is located in the SAS directory files in the CAF. The directory contains file metadata information specific to Windows files. For example on the UNIX platform, there does not exist a hidden flag. This platform specific information needs to be transmitted to the client to fool the streamed application into believing that the application data is located natively on the client machine with all the associated file metadata intact. If SAS is to be used to support streaming of UNIX or MacOS applications, file metadata specific to those systems will need to be recorded in the SAS directory.

[0366] Lastly, the format of the file names itself is OS dependent. Applications running on the Windows environment inherit the old MSDOS 8.3 file name format. To support this properly, the format of the SAS Directory file in CAF requires an additional 8.3 field to store this information. This field is not needed in other operating systems like UNIX or MacOS.

[0367] Device Driver Versus File System Paradigm

[0368] Referring to FIGS. 29 and 30, the SAS client Prototype is implemented using the 'device driver' paradigm. One of the advantages of the device driver approach is that the caching of the sector blocks is simpler. In the device driver approach, the client cache manager 2902 only needs to track sector numbers in its cache 2903. In comparison with the 'file system' paradigm, more complex data structure are required by the client cache manager 3002 to track a subset of a file that is cached 3003 on a client machine. This makes the device driver paradigm easier to implement.

[0369] On the other hand, there are many drawbacks to the device driver paradigm. On the Windows system, the device driver approach has a problem supporting large numbers of applications. This is due to the phantom limitation on the number of assignable drive letters available in a Windows system (26 letters);

[0370] and the fact that each application needs to be located on its own device. Note that having multiple applications on a device is possible, but then the server needs to maintain an exponential number of devices that support all possible combinations of applications. This is too costly to maintain on the server.

[0371] Another problem with the device driver approach is that the device driver operates at the disk sector level. This is a much lower level than operating at the file level in the file system approach. The device driver does not know anything about files. Thus, the device driver cannot easily interact with the file level issues.

[0372] For example, spoofing files and interacting with the OS file cache is nearly impossible with the device driver approach. Both spoofing files and interacting with the OS buffer cache are needed to get higher performance. In addition, operating at the file system level leads to optimizing the file system to better suit this approach of running applications. For instance, typical file systems do logging and make multiple disk sector requests at a time. These are not needed in this approach and are actually detrimental to the performance. When operating at the device driver level, not much can be done about that. Also, operating at the file system level helps in optimizing the protocol between the client and the server.

[0373] Implementation in the Prototype

[0374] The prototype has been implemented and tested successfully on the Windows and Linux distributed system. The prototype is implemented using the device driver paradigm as described above. The exact procedure for streaming application data is described next.

[0375] First of all, the prototype server is started on either the Windows-based or Linux-based system. The server creates a large local file mimicking large local disk images. Once the disk images are prepared, it listens to TCP/IP ports for any disk sector read or write requests.

[0376] Implementation of SAS Builder

[0377] The SAS Builder has been implemented on the Windows-based platform. A preliminary Streamed Application Set file has been created for real-world applications like Adobe Photoshop. A simple extractor program has been developed to extract the SAS data on a pristine machine without the application installed locally. Once the extractor program is run on the SAS, the application runs as if it was installed locally on that machine. This process verifies the correctness of the SAS Building process.

FORMAT OF STREAMED APPLICATION SET (SAS)

[0378] Functionality

[0379] The streamed application set (SAS), illustrated in FIG. 28, is a data set associated with an application suitable for streaming over the network. The SAS is generated by the

SAS Builder program. The program converts locally installable applications into the SAS. This section describes the format of the SAS.

[0380] Note: Fields greater than a single byte are stored in little-endian format. The Stream Application Set (SAS) file size is limited to 2A64 bytes. The files in the CAF section are laid out in the same order as its corresponding entries in the SOFT table.

[0381] Data Type Definitions

[0382] The format of the SAS consists of four sections: header, Root Version Table (RVT), Size Offset File Table (SOFT), and Concatenation Application File (CAF) sections.

[0383] 1. Header section

[0384] MagicNumber [4 bytes]: Magic number identifying the file content with the SAS.

[0385] ESSVersion [4 bytes]: Version number of the SAS file format.

[0386] AppID [16 bytes]: A unique application ID for this application. This field must match the AppID located in the AppInstallBlock. Window Guidgen API is used to create this identifier.

[0387] Flags [4 bytes]: Flags pertaining to SAS.

[0388] Reserved [32 bytes]: Reserved spaces for future.

[0389] RVToffset [8 bytes]: Byte offset into the start of the RVT section.

[0390] RVTsize [8 bytes]: Byte size of the RVT section.

[0391] SOFToffset [8 bytes]: Byte offset into the start of the SOFT section.

[0392] SOFTsize [8 bytes]: Byte size of the SOFT section.

[0393] CAFoffset [8 bytes]: Byte offset into the start of the CAF section.

[0394] CAFsize [8 bytes]: Byte size of the CAF section.

[0395] VendorNamesAnsi [1 byte]: 0 if the vendor name is in Unicode format. 1 if the vendor name is in ANSI format.

[0396] VendorNameLength [4 bytes]: Byte length of the vendor name.

[0397] VendorName [X bytes]: Name of the software vendor who created this application. e.g., "Microsoft". Null-terminated.

[0398] AppBaseNamesAnsi [1 byte]: 0 if the vendor name is in Unicode format. 1 if the vendor name is in ANSI format.

[0399] AppBaseNameLength [4 bytes]: Byte length of the application base name.

[0400] AppBaseName [X bytes]: Base name of the application. e.g., "Word 2000". Null-terminated.

[0401] MessagesAnsi [1 byte]: 0 if the vendor name is in Unicode format. 1 if the vendor name is in ANSI format.

[0402] MessageLength [4 bytes]: Byte length of the message text.

[0403] Message [X bytes]: Message text. Null-terminated.

[0404] 2. Root Version Table (RVT) section

[0405] The Root version entries are ordered in a decreasing value according to their file numbers. The Builder generates unique file numbers within each SAS in a monotonically increasing value. So larger root file numbers imply later versions of the same application. The latest root version is located at the top of the section to allow the SAS Server easy access to the data associated with the latest root version.

[0406] NumberEntries [4 bytes]: Number of patch versions contained in this SAS. The number indicates the number of entries in the Root Version Table (RVT).

[0407] Root Version structure: (variable number of entries)

[0408] VersionNumber [4 bytes]: Version number of the root directory.

[0409] FileName [4 bytes]: File number of the root directory.

[0410] VersionNamesAnsi [1 byte]: 0 if the vendor name is in Unicode format. 1 if the vendor name is in ANSI format.

[0411] VersionNameLength [4 bytes]: Byte length of the version name

[0412] VersionName [X bytes]: Application version name. e.g., "SP 1".

[0413] Metadata [32 bytes]: See SAS FS Directory for format of the metadata.

[0414] 3. Size Offset File Table (SOFT) section

[0415] The SOFT table contains information to locate specific files in the CAF section. The entries are ordered according to the file number starting from 0 to NumberFiles-1. The start of the SOFT table is aligned to eight-byte boundaries for faster access.

[0416] SOFT entry structure: (variable number of entries)

[0417] Offset [8 bytes]: Byte offset into CAF of the start of this file.

[0418] Size [8 bytes]: Byte size of this file. The file is located from address Offset to Offset+Size.

[0419] 4. Concatenation Application File (CAF) section

[0420] CAF is a concatenation of all file or directory data into a single data structure. Each piece of data can be a regular file, an AppInstallBlock, an SAS FS directory file, or an icon file.

[0421] a. Regular Files

[0422] FileData [X bytes]: Content of a regular file

[0423] b. AppInstallBlock (See AppInstallBlock section for detailed format) A simplified description of the AppInstallBlock is listed here. The exact detail of the individual fields in the AppInstallBlock are described later.

[0424] Header section [X bytes]: Header for AppInstallBlock containing information to identify this AppInstallBlock.

[0425] Files section [X bytes]: Section containing file to be copied or spoofed.

[0426] AddVariable section [X bytes]: Section containing system variables to be added.

[0427] RemoveVariable section [X bytes]: Section containing system variables to be removed.

[0428] Prefetch section [X bytes]: Section containing pointers to file blocks to be prefetched to the client.

[0429] Profile section [X bytes]: Section containing profile data.

[0430] Comment section [X bytes]: Section containing comments about AppInstallBlock.

[0431] Code section [X bytes]: Section containing application-specific code needed to prepare local machine for streaming this application

[0432] LicenseAgreement section [X bytes]: Section containing licensing agreement message.

[0433] c. SAS Directory

[0434] An SAS Directory contains information about the subdirectories and files located within this directory. This information is used to store metadata information related to the files associated with the streamed application. This data is used to fool the application into thinking that it is running locally on a machine when most of the data is resided elsewhere.

[0435] The SAS directory contains information about files in its directory. The information includes file number, names, and metadata associated with the files.

[0436] MagicNumber [4 bytes]: Magic number for SAS directory file.

[0437] ParentFileID [16+4 bytes]: AppID+FileNumber of the parent directory. AppID is set to 0 if the directory is the root.

[0438] SelfFileID [16+4 bytes]: AppID+FileNumber of this directory.

[0439] NumFiles [4 bytes]: Number of files in the directory.

[0440] Variable-Sized File Entry:

[0441] UsedFlag [1 byte]: 1 for used, 0 for unused.

[0442] ShortLen [1 byte]: Length of short file name.

[0443] LongLen [2 byte]: Length of long file name.

[0444] NameHash [4 bytes]: Hash value of the short file name for quick lookup without comparing whole string.

[0445] ShortName [24 bytes]: 8.3 short file name in Unicode. Not null-terminated.

[0446] FileID [16+4 bytes]: AppID+FileNumber of each file in this directory.

[0447] Metadata [32 bytes]: The metadata consists of file byte size (8 bytes), file creation time (8 bytes), file modified time (8 bytes), attribute flags (4 bytes), SAS flags (4 bytes). The bits of the attribute flags have the following meaning:

[0448] Bit 0: Read-only—Set if file is read-only

[0449] Bit 1: Hidden—Set if file is hidden from user

[0450] Bit 2: Directory—Set if the file is an SAS Directory

[0451] Bit 3: Archive—Set if the file is an archive

[0452] Bit 4: Normal—Set if the file is normal

[0453] Bit 5: System—Set if the file is a system file

[0454] Bit 6: Temporary—Set if the file is temporary

[0455] The bits of the SAS flags have the following meaning:

[0456] Bit 0: ForceUpgrade—Used only on root file. Set if client is forced to upgrade to this particular version if the current root version on the client is older.

[0457] Bit 1: RequireAccessToken—Set if file require access token before client can read it.

[0458] Bit 2: Read-only—Set if the file is read-only

[0459] LongName [X bytes]: Long filename in Unicode format with null-termination character.

[0460] d. Icon files

[0461] IconFileData [X bytes]: Content of an icon file.

FORMAT OF APPINSTALLBLOCK

[0462] Functionality

[0463] With respect to FIGS. 31a-31h, the AppInstallBlock is a block of code and data associated with a particular application. This AppInstallBlock contains the information needed to by the SAS client to initialize the client machine before the streamed application is used for the first time. It also contains optional profiling data for increasing the runtime performance of that streamed application.

[0464] The AppInstallBlock is created offline by the SAS Builder program. First of all, the Builder monitors the installation process of a local version of the application installation program and records changes to the system. This includes any environment variables added or removed from the system 3103, and any files added or modified in the system directories 3102. Files added to the application specific directory are not recorded in the AppInstallBlock to reduce the amount of time needed to send the AppInstallBlock to the SAS client. Secondly, the Builder profiles the application to obtain the list of critical pages needed to run the application initially and an initial page reference sequence of the pages accessed during a sample run of the

application. The AppInstallBlock contains an optional application-specific initialization code 3107. This code is needed when the default initialization procedure is insufficient to setup the local machine environment for that particular application.

[0465] The AppInstallBlock and the runtime data are packaged into the SAS by the Builder and then uploaded to the application server. After the SAS client is subscribed to an application and before the application is run for the first time, the AppInstallBlock is sent by the server to the client. The SAS client invokes the default initialization procedure and the optional application-specific initialization code 3107. Together, the default and the application-specific initialization procedure process the data in the AppInstallBlock to make the machine ready for streaming that particular application.

[0466] Data type definitions

[0467] The AppInstallBlock is divided into the following sections: header section 3101, variable section 3103, file section 3102, profile section 3105, prefetch section 3104, comment section 3106, and code section 3107. The header section 3101 contains general information about the AppInstallBlock. The information includes the total byte size and an index table containing size and offset into other sections. In a Windows version, the variable section consists of two registry tree structures to specify the registry entries added or removed from the OS environment. The file section 3102 is a tree structure consisting of the files copied to C drive during the application installation. The profile section 3105 contains the initial set of block reference sequences during Builder profiling of the application. The prefetch section 3104 consists of a subset of profiled blocks used by the Builder as a hint to the SAS client to prefetch initially. The comment section 3106 is used to inform the SAS client user of any relevant information about the application installation. Finally, the code section 3107 contains an optional program tailored for any application-specific installation not covered by the default streamed application installation procedure. In Windows version, the code section contains a Windows DLL. The following is a detailed description of each fields of the AppInstallBlock.

[0468] Note: Little endian format is used for all the fields spanning more than 1 byte. Also, BlockNumber specifies blocks of 4 K byte size.

[0469] Header Section

[0470] The header section 3103 contains the basic information about this AppInstallBlock. This includes the versioning information, application identification, and index into other sections of the file.

[0471] Core Header Structure

[0472] AibVersion [4 bytes]: Magic number or appInstallBlock version number (which identifies the version of the appInstallBlock structure rather than the contents).

[0473] AppId [16 bytes]: this is an application identifier unique for each application. On Windows, this identifier is the GUID generated from the 'guidgen' program. AppId for Word on Win98 will be different from Word on WinNT if it turns out that Word binaries are different between NT and 98.

[0474] VersionNo [4 bytes]: Version number. This allows us to inform the client that the appInstallBlock has changed for a particular appld. This is useful for changes to the AppInstallBlock due to minor patch upgrades in the application.

[0475] ClientOSBitMap [4 bytes]: Client OS supported bitmap or ID: for Win2K, Win98, WinNT (and generally for other and multiple OSs).

[0476] ClientOSServicePack [4 bytes]: For optional storage of the service pack level of the OS for which this appInstallBlock has been created. Note that when this field is set, the multiple OS bits in the above field ClientOSBitMap are not used.

[0477] Flags [4 bytes]: Flags pertaining to AppInstallBlock

[0478] Bit 0: Reboot—If set, the SAS client needs to reboot the machine after installing the AppInstallBlock on the client machine.

[0479] Bit 1: Unicode—If set, the string characters are 2 bytes wide instead of 1 byte.

[0480] HeaderSize [2 bytes]: Total size in bytes of the header section.

[0481] Reserved [32 bytes]: Reserved spaces for future.

[0482] NumberOfSections [1 byte]: Number of sections in the index table.

[0483] This determines the number of entries in the index table structure described below:

[0484] Index Table Structure: (variable number of entries)

[0485] SectionType [1 bytes]: The type of data described in section. 0=file section, 1=variable section, 2=prefetch section, 3=profile section, 4=comment section, 5=code section.

[0486] SectionOffset [4 bytes]: The offset from the beginning of the file indicates the beginning of section.

[0487] SectionSize [4 bytes]: The size in bytes of section.

[0488] Variable Structure

[0489] ApplicationNameIsAnsi [1 byte]: 1 if ansi, 0 if Unicode.

[0490] ApplicationNameLength [4 bytes]: Byte size of the application name

[0491] ApplicationName [X bytes]: Null terminating name of the application

[0492] 2. File Section

[0493] The file section 3102 contains a subset of the list of files needed by the application to run properly. This section does not enumerate files located in the standard application program directory. It consists of information about files copied into an 'unusual' directory during the installation of an application. If the file content is small (typically less than 1 MB), the file is copied to the client machine. Otherwise, the file is relocated to the standard program directory suitable for streaming. The file section data is a list of trees

stored in a contiguous sequence of address spaces according to the pre-order traversal of the trees. A node in the tree can correspond to one or more levels of directories. A parent-child node pair is combined into a single node if the parent node has only a single child. Parsing the tree from the root of the tree to a leaf node results in a fully legal Windows pathname including the drive letter. Each entry of the node in the tree consists of the following structure:

[0494] Directory Structure: (variable number of entries)

- [0495] Flags [4 byte]:** Bit 0 is set if this entry is a directory
- [0496] NumberOfChildren [2 bytes]:** Number of nodes in this directory
- [0497] DirectoryNameLength [4 bytes]:** Length of the directory name
- [0498] DirectoryName [X bytes]:** Null terminating directory name

[0499] Leaf Structure: (variable number of entries)

- [0500] Flags [4 byte]:** Bit 1 is set to 1 if this entry is a spoof or copied file name
- [0501] FileVersion [8 bytes]:** Version of the file GetFileVersionInfo() if the file is win32 file image. Need variable file version size returned by GetFileVersionInfoSize(). Otherwise use file size or file modified time to compare which file is the later version.
- [0502] FileNameLength [4 bytes]:** Byte size of the file name
- [0503] FileName [X bytes]:** Null terminating file name
- [0504] DataLength [4 bytes]:** Byte size of the data. If spoof file, then data is the string of the spoof directory. If copied file, then data is the content of the file
- [0505] Data [X bytes]:** Either the spoof file name or the content of the copied file

[0506] 3. Add Variable and Remove Variable Sections

[0507] The add and remove variable sections 3103 contain the system variable changes needed to run the application. In a Windows system, each section consists of several number of registry subtrees. Each tree is stored in a contiguous sequence of address spaces according to the pre-order traversal of the tree. A node in the tree can correspond to one or more levels of directory in the registry. A parent-child node pair is combined into a single node if the parent node has only a single child. Parsing the tree from the root of the tree to a leaf node results in a fully legal key name. The order of the trees is shown here.

[0508] a. Registry Subsection:

- [0509] 1. "HKCR":** HKEY_CLASSES_ROOT
- [0510] 2. "HKCU":** HKEY_CURRENT_USER
- [0511] 3. "HKLM":** HKEY_LOCAL_MACHINE
- [0512] 4. "HKUS":** HKEY_USERS
- [0513] 5. "HKCC":** HKEY_CURRENT_CONFIG

[0514] Tree Structure: (5 entries)

- [0515] ExistFlag [1 byte]:** Set to 1 if this tree exist, 0 otherwise.
- [0516] Key or Value Structure entries [X bytes]:** Serialization of the tree into variable number key or value structures described below.
- [0517] Key Structure: (variable number of entries)**
 - [0518] KeyFlag [1 byte]:** Set to 1 if this entry is a key or 0 if it's a value structure
 - [0519] NumberOfSubchild [4 bytes]:** Number of subkeys and values in this key directory
 - [0520] KeyNameLength [4 bytes]:** Byte size of the key name
 - [0521] KeyName [X bytes]:** Null terminating key name

[0522] Value Structure: (variable number of entries)

- [0523] KeyFlag [1 byte]:** Set to 1 if this entry is a key or 0 if it's a value structure
- [0524] ValueType [4 byte]:** Type of values from the Win32 API function RegQueryValueEx(): REG_SZ, REG_BINARY, REG_DWORD, REG_LINK, REG_NONE, etc . . .
- [0525] ValueNameLength [4 bytes]:** Byte size of the value name
- [0526] ValueName [X bytes]:** Null terminating value name
- [0527] ValueDataLength [4 bytes]:** Byte size of the value data
- [0528] ValueData [X bytes]:** Value of the Data

[0529] In addition to registry changes, an installation in a Windows system may involve changes to the ini files. The following structure is used to communicate the ini file changes needed to be done on the SAS client machine. The ini entries are appended to the end of the variable section after the five registry trees are enumerated.

[0530] b. INI Subsection:

- [0531] NumFiles [4 bytes]:** Number of INI files modified.

[0532] File Structure: (variable number of entries)

- [0533] FileNameLength [4 bytes]:** Byte length of the file name
- [0534] FileName [X bytes]:** Name of the INI file
- [0535] NumSection [4 bytes]:** Number of sections with the changes

[0536] Section Structure: (variable number of entries)

- [0537] SectionNameLength [4 bytes]:** Byte length of the section name
- [0538] SectionName [X bytes]:** Section name of an INI file
- [0539] NumValues [4 bytes]:** Number of values in this section

[0540] Value Structure: (variable number of entries)**[0541] ValueLength [4 bytes]:** Byte length of the value data**[0542] ValueData [X bytes]:** Content of the value data**[0543] 4. Prefetch Section**

[0544] The prefetch section **3104** contains a list of file blocks. The Builder profiler determines the set of file blocks critical for the initial run of the application. This data includes the code to start and terminate the application. It includes the file blocks containing frequently used commands. For example, opening and saving of documents are frequently used commands and should be prefetched if possible. Another type of block to include in the prefetch section are the blocks associated with frequently accessed directories and file metadata in this directory. The prefetch section is divided into two subsections. One part contains the critical blocks that are used during startup of the streamed application. The second part consists of the blocks accessed for common user operations like opening and saving of document. The format of the data is described below:

[0545] a. Critical Block Subsection:**[0546] NumCriticalBlocks [4 bytes]:** Number of critical blocks.**[0547] Block Structure: (variable number of entries)****[0548] FileNumber [4 bytes]:** File Number of the file containing the block to prefetch**[0549] BlockNumber [4 bytes]:** Block Number of the file block to prefetch**[0550] b. Common Block Subsection:****[0551] NumCommonBlocks [4 bytes]:** Number of critical blocks.**[0552] Block Structure: (variable number of entries)****[0553] FileNumber [4 bytes]:** File Number of the file containing the block to prefetch**[0554] BlockNumber [4 bytes]:** Block Number of the file block to prefetch**[0555] 5. Profile Section**

[0556] The profile section **3105** consists of a reference sequence of file blocks accessed by the application at runtime. Conceptually, the profile data is a two dimensional matrix. Each entry [row, column] of the matrix is the frequency, a block row is followed by a block column. In any realistic applications of fair size, this matrix is very large and sparse. The proper data structure must be selected to store this sparse matrix efficiently in required storage space and minimize the overhead in accessing this data structure access.

[0557] The section is constructed from two basic structures: row and column structures. Each row structure is followed by N column structures specified in the Number-Columns field. Note that this is an optional section. But with appropriate profile data, the SAS client prefetcher performance can be increased.

[0558] Row Structure: (variable number of entries)**[0559] FileNumber [4 bytes]:** File Number of the row block**[0560] BlockNumber [4 bytes]:** Block Number of the row block**[0561] NumberColumns [4 bytes]:** number of blocks that follows this block. This field determines the number of column structures following this field.**[0562] Column Structure: (variable number of entries)****[0563] FileNumber [4 bytes]:** File Number of the column block**[0564] BlockNumber [4 bytes]:** Block Number of the column block**[0565] Frequency [4 bytes]:** frequency the row block is followed by column block**[0566] 6. Comment Section**

[0567] The comment section **3106** is used by the Builder to describe this AppInstallBlock in more detail.

[0568] CommentLengthAnsi [1 byte]: 1 if string is ansi, 0 if Unicode format.**[0569] CommentLength [4 bytes]:** Byte size of the comment string**[0570] Comment [X bytes]:** Null terminating comment string**[0571] 7. Code Section**

[0572] The code section **3107** consists of the application-specific initialization code needed to run on the SAS client to setup the client machine for this particular application. This section may be empty if the default initialization procedure in the SAS client is able to setup the client machine without requiring any application-specific instructions. On the Windows system, the code is a DLL file containing two exported function calls: Install(), Uninstall(). The SAS client loads the DLL and invokes the appropriate function calls.

[0573] CodeLength [4 bytes]: Byte size of the code**[0574] Code [X bytes]:** Binary file containing the application-specific initialization code. On Windows, this is just a DLL file.**[0575] 8. LicenseAgreement Section**

[0576] The Builder creates the license agreement section **3108**. The SAS client displays the license agreement text to the end-user before the application is started for the first time. The end-user must agree to all licensing agreement set by the software vendor in order to use the application.

[0577] LicenseTextisAnsi [1 byte]: 1 if ansi, 0 if Unicode format.**[0578] LicenseTextLength [4 bytes]:** Byte size of the license text o LicenseAgreement [X bytes]: Null terminating license agreement string

CLIENT INSTALLATION AND EXECUTION OF STREAMED APPLICATIONS

[0579] Summary

[0580] This section describes the process of installing and uninstalling streamed application on the client machine. With respect to FIG. 32, the lifecycle of the Application Install Block is shown. The Application Stream Builder 3202 takes original application files 3201 and produces a corresponding Application Install Block and Stream Application Set 3203. These two files get installed onto the application servers 3206. On the right side of the drawing, it shows how either the administrator or the user can subscribe to the application from either the client computer 3208 or an administration computer 3207. Once the user logons onto the client computer 3208, the license and the AIB 3203 are acquired from the license 3205 and application servers 3206, respectively.

[0581] The following are features of a preferred embodiment of the invention:

[0582] 1. The streamed application installation process installs just the description of the application, not the total content of the application. After installing such description, the client system looks and feels similar to having installed the same app using a non-streamed method. This has the following benefits:

[0583] a. Takes a very small fraction of the time to install the application.

[0584] b. Takes a very small fraction of the disk space.

[0585] c. Client does not have to wait for the entire application to be downloaded. This is particularly important to users with slow network connections.

[0586] The application description is subsequently un-installed without requiring deleting the total contents of the application. This has the benefit that it takes a very small fraction of the time to uninstall the application.

[0587] 2. Enhancing streamed application's performance by:

[0588] a. Copying small portions of application's code and data (pages) that are critical to performance.

[0589] b. Providing client with the initial profile data that can be used to perform pre-fetching.

[0590] This has the following benefits:

[0591] 1. User experiences smooth and predictable application launch.

[0592] 2. Scalability of Application servers increases by reducing the number of client connections.

[0593] 3. An administrator can arrange applications to be installed automatically on client computers. Administrator can also arrange the installation on various client computers simultaneously without being physically present on each client computer. This has the following benefits:

[0594] a. Users are not burdened with the process of installing streamed applications.

[0595] b. Reduced administration expense.

[0596] Overview of Components Relevant to the Install Process

[0597] Subscription Server 3204: allows users to create accounts & to rent.

[0598] License Server 3205: authenticates users & determines licensing rights to applications.

[0599] Application Server 3206: provides application bits to licensed users securely & efficiently.

[0600] Application Install Manager—a component installed on the streaming client that is responsible for installing and uninstalling streamed applications.

[0601] Application Install Block (AIB) 3203—a representation of what gets installed on the client machine when a streamed application is installed. It contains portions of the application that are responsible for registering the application with the client operating system and other data that enhances the execution of streamed application.

[0602] Application Stream Builder 3202—preprocesses apps & prepares files to be installed on Application Server and data, such as AIB, to be installed by Client Application Installer.

[0603] Stream Application Set 3203—a method of representing the total content of the application in a format that is optimal for streaming.

[0604] Client Streaming File System—integrates client exec with paging from a special file system backed by remote network-accessed server-based store

[0605] Application Install Block (AIB)

[0606] Installing and un-installing a stream application requires an understanding of what AIB is and how it gets manipulated by the various components in the overall streaming system. AIB is physically represented as a data file with various different sections. Its contents include:

[0607] Streamed application name and identification number.

[0608] Software License Agreement.

[0609] Registry spoof set.

[0610] File spoof set.

[0611] Small number of application pages—initial cache contents.

[0612] Application Profile Data.

[0613] AIB Lifecycle

[0614] The following describes the AIB lifecycle:

[0615] 1. Using the process described in the section above concerning converting apps for stream delivery and subsequent execution, an application install block is created by the Application Stream Builder. Initially, there will be one AIB per application, however, as the application evolves via patches and service packs, new AIBs may need to be generated.

- [0616] 2. Using a process described in the section above regarding server-side performance optimization, AIB will get hosted by the application servers.
- [0617] 3. "Subscribing" the application by communicating with the subscription server. Subscribing to an application requires a valid account with the ASP. Either the user or an administrator acting on the user's behalf can subscribe the application. In addition, the application can be subscribed to from any computer on the Internet, not just the client machine where the application will be eventually installed. This allows an administrator to subscribe applications for a group of users without worrying about individual client machines.
- [0618] 4. The client machine acquires the license for the application from the license server. If the application was subscribed from the client machine itself, this step will happen automatically after subscribing to the application. If the subscription happened from a different machine, e.g., the administrator's machine, this step will happen when the user logs on the client machine. As an acknowledgment of having a valid license, the license server gives the client an encrypted access token.
- [0619] 5. Fetch the contents of AIB from the application server. This step is transparent and happens immediately after the preceding step. Since application server requires the client to possess a valid access token, it ensures that only subscribed and licensed users can install the streamed application.
- [0620] 6. The Application Install Manager (AIM) performs the act of installing the application information, as specified by the AIB, on the client system.
- [0621] Installing a Streamed Application
- [0622] AIM downloads AIB from the application server and takes the necessary steps in installing the application description on the client system. It extracts pieces of information from AIB and sends messages to various other components (described later) to perform the installation. AIM also creates an Install-Log that can be used when un-installing the streamed application.
- [0623] 1. Display a license agreement to the user and wait for the user to agree to it.
- [0624] 2. Extract File Spoof Data and communicate that to the Client File Spoofer. The list of files being spoofed will be recorded in the Install-Log.
- [0625] 3. Extract Registry Spoof Data and communicate that to the Client Registry Spoofer. The list of Registries being spoofed will be recorded in the Install-Log.
- [0626] 4. Extract Initial Cache Content and communicate that to the Client Prefetch Unit.
- [0627] 5. Extract Profile Data and communicate that to the Client Prefetch Unit.
- [0628] 6. Save the Install-Log in persistent storage.
- [0629] Un-Installing a Streamed Application
- [0630] Un-installation process relies on the Install-Log to know what specific items to un-install. Following steps are performed when un-installing and application:
- [0631] 1. Communicate with the Client Registry Spoofer to remove all registries being spoofed for the application being un-installed.
- [0632] 2. Communicate with the Client File Spoofer to disable all files being spoofed for the application being un-installed.
- [0633] 3. Communicate with the Client Prefetch Unit to remove all Profile Data for the application being un-installed.
- [0634] 4. Communicate with the Client Cache Manager to remove all pages being cached for the application being un-installed.
- [0635] 5. Delete the Install-Log.
- [0636] Client File Spoofer
- [0637] A file spoofer component is installed on the client machine and is responsible for redirecting file accesses from a local file system to the streaming file system. The spoofer operates on a file spoof database that is stored persistently on the client system; it contains a number of file maps with following format:
- [0638] [Original path of a local file]↔[New path of a file on streaming drive]
- [0639] Where "↔" indicates a bidirectional mapping between the two sides of the relationship shown.
- [0640] When a streamed application is installed, the list of new files to spoof (found in AIB) is added to the file spoof database. Similarly, when a streamed application is un-installed, a list of files being spoofed for that application is removed from the file spoof database.
- [0641] On clients running the Windows 2000 Operating System, the file spoofer is a kernel-mode driver and the spoof database is stored in the registry.
- [0642] Client Registry Spoofer
- [0643] The Registry Spoofer intercepts all registry calls being made on the client system and re-directs calls manipulating certain registries to an alternate path. Effectively, it is mapping the original registry to an alternate registry transparently. Similar to the client file spoofer, the registry spoofer operates on a registry spoof database consisting entries old/new registry paths. The database must be stored in persistent storage.
- [0644] When a streamed application is installed, the list of new registries to spoof (found in AIB) is added to the registry spoof database. Upon un-installation of a streamed application, its list of spoofed registries is removed from the registry spoof database.
- [0645] On clients running the Windows 2000 Operating System, the registry spoofer is a kernel-mode driver and the registry spoof database is stored in the registry.

[0646] Client Prefetch Unit

[0647] In a streaming system, it is often a problem that the initial invocation of the application takes a lot of time because the necessary application pages are not resent on the client system when needed. A key aspect of the client install is that by using a client prefetch unit, a system in accordance with the present invention significantly reduces the performance hit associated with fetching. The Client Prefetch Unit performs two main tasks:

[0648] 1. Populate Initial Cache Content,

[0649] 2. Prefetch Application Pages.

[0650] Initial Cache Content

[0651] The Application Stream Builder determines the set of pages critical for the initial invocation and packages them as part of the AIB. These pages, also known as initial cache content, include:

[0652] Pages required to start and stop the application,

[0653] Contents of frequently accessed directories,

[0654] Application pages performing some of the most common operations within application. For example, if Microsoft Word is being streamed, these operations include: opening & saving document files & running a spell checker.

[0655] When the Stream Application is installed on the client, these pages are put into the client cache; later, when the streamed application is invoked, these pages will be present locally and network latency is avoided.

[0656] In preparing the Prefetch data, it is critical to manage the trade off of how many pages to put into AIB and what potential benefits it brings to the initial application launch. The more pages that are put into prefetch data, the smoother the initial application launch will be; however, since the AIB will get bigger (as a result of packing more pages in it), users will have to wait longer when installing the streamed application. In a preferred embodiment of the invention, the size of the AIB is limited to approximately 250 KB.

[0657] In an alternative embodiment of the invention the AIB initially includes only the page/file numbers and not the pages themselves. The client then goes through the page/file numbers and does paging requests to fetch the indicated pages from the server.

[0658] Prefetch Application Pages

[0659] When the streaming application executes, it will generate paging requests for pages that are not present in the client cache. The client cache manager must contact the application server and request the page in question. The invention takes advantage of this opportunity to also request additional pages that the application may need in the future. This not only reduces the number of connections to the application server, and overhead related to that, but also hides the latency of cache misses.

[0660] The application installation process plays a role in the pre-fetching by communicating the profile data present in the AIB to the Client Prefetch Unit when the application is installed. Upon un-installation, profile data for the particular application will be removed.

CACHING OF STREAMED APPLICATION PAGES WITHIN THE NETWORK**[0661] Summary**

[0662] This section describes how collaborative caching is employed to substantially improve the performance of a client server system in accordance with the other aspects of the present invention. Specifically, particular caching configurations and an intelligent way to combine these caching configurations are detailed.

[0663] Collaborative Caching Features

[0664] Using another client's cache to get required pages/packets (Peer Caching)

[0665] Using an intermediate proxy or node to get required pages/packets (Proxy Caching)

[0666] Using a broadcasting or multicasting mechanism to make a request (Multicast)

[0667] Using a packet based protocol to send requested pages/packets rather than a stream based one. (Packet Protocol)

[0668] Using concurrency to request a page through all three means (Peer Caching or Proxy Caching or the actual server) to improve performance (Concurrent Requesting).

[0669] Using heuristical algorithms to use all three ways to get the required pages (Smart Requesting).

[0670] These features have the following advantages:

[0671] These ideas potentially improve the performance of the client, i.e., they reduce the time a client takes to download a page (Client Performance).

[0672] These ideas improve the scalability of the server because the server gets fewer requests, i.e., requests which are fulfilled by a peer or a proxy don't get sent to the server. (Server Scalability)

[0673] These allow a local caching mechanism without needing any kind of modification of local proxy nodes or routers or even the servers. The peer-to-peer caching is achieved solely through the co-operation of two clients. (Client Only Implementation)

[0674] These ideas allow a client to potentially operate "offline" i.e., when it is not getting any responses from the server (Offline Client Operation).

[0675] These ideas allow the existing network bandwidth to be used more effectively and potentially reduce the dependency of applications on higher bandwidth (Optimal Use of Bandwidth).

[0676] These ideas when used in an appropriate configuration allow each client to require a smaller local cache but without substantially sacrificing the performance that you get by local caching. An example is when each client "specializes" in caching pages of a certain kind, e.g., a certain application. (Smaller Local Cache).

[0677] These ideas involve new interrelationships—peer-to-peer communication for cache accesses; or new configurations—collaborative caching. The reason this is called collaborative is because a group of clients can collaborate in caching pages that each of them needs.

[0678] Aspects of Collaborative Caching

[0679] 1. Peer Caching: A client X getting its pages from another client Y's local cache rather than its (X's) own or from the server seems to be a new idea. Major advantages: client performance, server scalability, client only implementation, offline client operation, optimal use of bandwidth, smaller local cache.

[0680] 2. Proxy Caching: The client getting its pages from an intermediate proxy which either serves the page from the local cache or from another intermediate proxy or the remote server (if none of the intermediate proxies has the page) is unique, at a minimum, for the pages of a streamed application. Major advantages: client performance, server scalability, offline client operation (to some extent), optimal use of bandwidth, smaller local cache.

[0681] 3. Multicast: Using multicasting (or selective broadcasting) considerably reduces peer-to-peer communication. For every cache request there is only one packet on the network and for every cache response there is potentially only one packet on the network in some configurations. This definitely helps reduce network congestion. Major advantages: client performance, server scalability, client only implementation, offline client operation, optimal use of bandwidth

[0682] 4. Packet Protocol: Because only datagram packets are used to request or respond to cache pages this saves the overhead of opening stream-based connections such as a TCP connection or an HTTP connection. Major advantages: client performance, client only implementation, offline client operation, and optimal use of bandwidth.

[0683] 5. Concurrent Requesting: If concurrent or intelligently staggered requests through all three means are issued to request a single page, the client will be able to receive the page through the fastest means possible for that particular situation. Major advantages: client performance, server scalability, offline client operation, and optimal use of bandwidth

[0684] 6. Smart Requesting: An adaptive or "smart" algorithm can be used to further enhance the overall performance of the client-server system. In this algorithm, the client uses the data of how past requests were processed to "une" new requests. For example, if the client's past requests were predominantly served by another client, i.e., Peer Caching worked, then for new page requests the client would first try to use Peer Caching, and wait some time before resorting to either Proxy Caching or direct server request. This wait time can again be calculated in an adaptive fashion. Major advantages: client performance, server scalability, client only implementation, offline client operation, and optimal use of bandwidth.

[0685] The concepts illustrated herein can be applied to many different problem areas. In all client-server implementations where a server is serving requests for static data, e.g., code pages of a streamed application or static HTML pages from a Website, the approaches taught herein can be applied to improve the overall client-server performance. Even if some of the protocols or configurations described in this document are not supported by the underlying network, it does not preclude the application of other ideas described herein that do not depend on such features. For example, if multicast (or selective broadcast) is not supported, ideas

such as Concurrent Requesting or Smart Requesting can still be used with respect to multiple servers instead of the combination of a server, peer, and proxy. Also the use of words like Multicast does not restrict the application of these ideas to multicast based protocols. These ideas can be used in all those cases where a multicast like mechanism, i.e., selective broadcasting is available. Also note that the description of these ideas in the context of LAN or intranet environment does not restrict their application to such environments. The ideas described here are applicable to any environment where peers and proxies, because of their network proximity, offer significant performance advantages by using Peer Caching or Proxy Caching over a simple client-server network communication. In that respect, the term LAN or local area network should be understood to mean more generally as a collection of nodes that can communicate with each other faster than with a node outside of that collection. No geographical or physical locality is implied in the use of the term local area network or LAN.

[0686] Peer Caching

[0687] Referring to FIG. 33, how multiple peers collaborate in caching pages that are required by some or all of them is shown.

[0688] The main elements shown are:

[0689] Client 13301 through Client 63306 in an Ethernet LAN 3310.

[0690] Router 1 and the local proxy serving as the Internet gateway 3307. Note that it does not matter whether Router 1 and the proxy are one computer or two different ones.

[0691] Other routers from router 2 through router N 3308 that are needed to connect the LAN 3310 to the Internet 3311.

[0692] A remote server 3309 (that is reachable only by going over the Internet 3311) that is serving the pages that the above mentioned clients need.

[0693] A cloud that symbolizes the complexity of the Internet 3311 and potentially long paths taken by packets.

[0694] Client 23302 needs a page that it does not find in its local cache. It then decides to use the mechanism of Peer Caching before attempting to get the page from the local proxy (or the actual server through the proxy). The actual sequence of events is as follows:

[0695] 1. Client 23302 sends a request for the page it needs. This request is sent as a multicast packet to a predetermined multicast address and port combination. Lets call this multicast address and port combination as M.

[0696] 2. The multicast packet is received by all the clients that have joined the group M. In this case all six clients have joined the group M.

[0697] 3. Client 53305 receives the request and it records the sender's, i.e., Client 2's 3302, address and port combination. Let's assume this address and port combination is A. Client 53305 processes the request and looks up the requested page in its own cache. It finds the page.

- [0698] 4. Client 53305 sends the page to address A (which belongs to Client 23302) as a packet.
- [0699] 5. Client 23302 receives the page it needs and hence does not need to request the server for the page.
- [0700] Proxy Caching
- [0701] With respect to FIG. 43, a transparent proxy and how clients use it to get pages is shown. Again the elements here are the same as in the previous figure:
- [0702] Client 13401 through Client 63406 in an Ethernet LAN 3410.
- [0703] Router 1 and the local proxy serving as the Internet gateway 3407. Note that it does not matter whether Router 1 and the proxy are one computer or two different ones.
- [0704] Other routers from router 2 through router N 3408 that are needed to connect the LAN 3410 to the Internet 3411.
- [0705] A remote server 3409 (that is reachable only by going over the Internet 3411) that is serving the pages that the above mentioned clients need.
- [0706] A cloud that symbolizes the complexity of the Internet 3411 and potentially long paths taken by packets.
- [0707] Assume Peer Caching is either not enabled or did not work for this case. When Client 23402 needs a page, it makes a request to the proxy 3407. The proxy 3407 finds the page in its local cache and returns it to Client 23402. Because of this, the request did not go to the remote server 3409 over the Internet 3411.
- [0708] Multicast and Packet Protocol within a LAN
- [0709] Referring to FIG. 35, the role played by multicast and unicast packets in Peer Caching is shown. The example of the drawing "Peer Caching" is used to explain multicast. Here Client 23502 has the IP address 10.0.0.2 and it opens port 3002 for sending and receiving packets. When Client 23502 needs a page and wants to use Peer Caching to get it, it forms a request and sends it to the multicast address and port 239.0.0.1:2001. All the other clients in the LAN 3508 that support Peer Caching have already joined the group 239.0.0.1:2001 so they all receive this packet.
- [0710] Client 53505 receives this packet and it records the sender address (10.0.0.2:3002 in this case). It looks up the requested page and finds it in its local cache. It sends the page as a response packet to the address 1.0.0.0.2:3002.
- [0711] Client 23502 receives this response packet since it was waiting at this port after sending the original multicast request. After ensuring the validity of the response, it retrieves the page it needs.
- [0712] Note that more than one client can respond to the original multicast request.
- [0713] However Client 23502 can discard all the later responses, since it has already received the page it needed.
- [0714] Concurrent Requesting—Proxy First
- [0715] With respect to FIG. 36, one particular case of how Concurrent Requesting is used is shown. This is a timeline of events that take place in the client. When a client first

needs a page, it does not know whether it is going to get any responses through Peer Caching or not. Hence it issues a request to the proxy (or the server through the proxy) as soon as it needs the page. Then it issues a request using the Peer Caching mechanism. If there is indeed a peer that can return the page requested, the peer presumably could return the page faster than the proxy or the server. If this happens, the client may decide to use Peer Caching mechanism before attempting to get the page from the proxy or the server. The timeline essentially describes the following sequence of events:

- [0716] 1. At time $t=0$, a page p is needed by the client 3601.
- [0717] 2. The client looks up its local cache, and it doesn't find page p .
- [0718] 3. At time $t=T_1$, it decides to send a request to the proxy to get the page 3602.
- [0719] 4. After a delay of amount D_p 3603, at time $t=T_2$ it also sends a request for the page p through the mechanism of Peer Caching 3604. Note that D_p 3603 can be zero, in which case $T_1=T_2$.
- [0720] 5. At time $t=T_3$, a response is received from another peer that contains the page p that this client needs 3606. Thus the response time of the Peer Caching mechanism is $R_p=T_3-T_2$ 3605.
- [0721] 6. At time $t=T_4$, a response from the proxy/server is received that contains the page p 3608. Hence the response time of the proxy/server is $R_s=T_4-T_1$ 3607.
- [0722] Note that since, $R_p < R_s$, the client will increase the weighting for Peer Caching in all of its future queries. That means it will decrease D_p , and if D_p is already zero, it will increase D_s (the delay before requesting proxy/server). On the other hand, if $R_p > R_s$ or if R_p were infinity, it will increase its weighting for proxy/server requesting. This is part of Smart Requesting that is explained elsewhere in this document.
- [0723] Concurrent Requesting—Peer Caching First
- [0724] Referring to FIG. 37, in contrast to the previous figure, the client has decided to use Peer Caching before requesting the proxy. So the sequence of events is as follows:
- [0725] 1. At time $t=0$, a page p is needed by the client 3701.
- [0726] 2. The client looks up its local cache, and it doesn't find page p .
- [0727] 3. At time $t=T_5$, it decides to send a request for the page p through the mechanism of Peer Caching 3702.
- [0728] 4. After a delay of amount D_s 3703, at time $t=T_6$ it also sends a request for the page p to the proxy/server. Note that D_s can be zero, in which case $T_5=T_6$.
- [0729] 5. At time $t=T_7$, a response is received from another peer that contains the page p that this client needs 3706. Thus the response time of the Peer Caching mechanism is $R_p=T_7-T_5$ 3705.

- [0730] 6. At time $t=T8$, a response from the proxy/server is received that contains the page p 3708. Hence the response time of the proxy/server is $R_s=T8-T6$ 3707.
- [0731] As described in the previous drawing, the client increases the weighting of Peer Caching even more because it got a response through Peer Caching long before it got a response from the proxy/server. As a result of the increases weighting the delay D_s is increased even more.
- [0732] Concurrent Requesting—Peer Caching Only
- [0733] With respect to FIG. 38, in contrast with FIG. 37, the client has increased D_s 3805 (the delay before requesting a proxy/server) so much, that if a page is received before the expiry of the delay D_s 3805, the client does not even make a request to the proxy/server. The shaded area 3806 shows the events that do not take place because of this.
- [0734] Client-Server System with Peer and Proxy Caching
- [0735] Referring to FIG. 39, a system level drawing that gives a system context for all the other figures and discussion in this document is shown. This drawing illustrates all three ways in which a client gets its page requests fulfilled. Note that:
- [0736] Client 23902 gets its page request fulfilled through Peer Caching, i.e., multicast request.
- [0737] Client 13901 gets its page request fulfilled through Proxy Caching, i.e., the proxy 3907 finds the page in its cache and returns it.
- [0738] Client 33903 has to go to the server 3909 over the Internet 3908 to get its page request fulfilled.
- [0739] Collaborative Caching Details
- [0740] In a typical client-server model, caching could be used to improve the performance of clients and scalability of servers. This caching could be:
- [0741] Local to the client where the client itself locally stores the pages it had received from the server in the past. Then the client would not need to request the proxy/server for any page that resides in the local cache as long as the locally cached page is "valid" from the server point of view.
- [0742] On a proxy node that can be any node along the path taken by a packet that goes from the client to the server. The closer this proxy node is to the client the more improvement in the performance you get.
- [0743] On a peer node, that is on another client. In this case, the two clients (the requesting client as well as the serving client) are on the same LAN or intranet, so that the travel time of a packet between the two nodes is considerably smaller as compared to the travel time of the packet from one of the clients to the server.
- [0744] As far as caching is concerned, this section details the new ideas of Peer Caching and Proxy Caching. In addition, it also details the new ideas of Concurrent Requesting and Smart Requesting. The preferred approaches for implementing these ideas are also described here and these are Multicast and Packet Protocol.
- [0745] The idea of Peer Caching is nothing but a client X taking advantage of the fact that a peer, e.g., say another client Y, on its LAN had, in the past, requested a page that X is going to request from its server. If the peer Y has that page cached locally on its machine, then X could theoretically get it much faster from Y than getting it from the server itself. If an efficient mechanism is provided for the two clients X and Y to collaborate on this kind of cache access, then that will offer many advantages such as: Client Performance, Server Scalability, Client Only Implementation, Offline Client Operation, Optimal Use of Bandwidth, Smaller Local Cache. Note that two clients were considered only as an example, the idea of Peer Caching is applicable to any number of peers on a LAN.
- [0746] The idea of Multicast is to use the multicast protocol in the client making a Peer Caching request. Multicast can be briefly described as "selective broadcasting"—similar to radio. A radio transmitter transmits "information" on a chosen frequency, and any receiver (reachable by the transmitter, of course) can receive that information by tuning to that frequency. In the realm of multicast, the equivalent of a radio frequency is a multicast or class D IP address and port. Any node on the net can send datagram packets to a multicast IP address+port. Another node on the net can "join" that IP address+port (which is analogous to tuning to a radio frequency), and receive those packets. That node can also "leave" the IP address+port and thereby stop receiving multicast packets on that IP address+port.
- [0747] Note that multicast is based on IP (Internet Protocol) and is vendor neutral. Also, it is typically available on the Ethernet LAN and, if routers supported it, it can also go beyond the LAN. If all the routers involved in a node's connection to the Internet backbone supported multicast routing, multicast packets theoretically could go to the whole Internet except the parts of the Internet that do not support multicast routing.
- [0748] The use of multicast allows a client to not have to maintain a directory of peers that can serve its page requests. Also because of multicast there is only one packet per page request. Any peer that receives the request could potentially serve that request, so by using a multicast based request there are multiple potential servers created for a page request but only one physical packet on the network.
- [0749] This contributes substantially in reducing network bandwidth, but at the same time increasing peer accessibility to all the peers. When implemented properly, the packet traffic due to Peer Caching will be proportional to the number of clients on the network participating in Peer Caching.
- [0750] An idea related to Multicast is Packet Protocol. Note that Multicast itself is a packet-based protocol as opposed to connection based. The idea of Peer Caching here is described using Multicast and Packet Protocol. The Peer Caching request is sent as a multicast request and the response from a peer to such a request is also sent as a packet (not necessarily a multicast packet). Sending packets is much faster than sending data through a connection-based protocol such as TCP/IP, although using packet-based protocol is not as reliable as using connection-based one. The lack of reliability in Packet Protocol is acceptable since Peer Caching is used only to improve overall performance of the Client-Server system rather than as a primary mechanism for a client to get its pages. The underlying assumption made here is that a client could always get its pages from the server, if Peer Caching or Proxy Caching does not work for any reason.

[0751] The ideas of Concurrent Requesting and Smart Requesting describe how Peer Caching, Proxy Caching and client-server access could be combined in an intelligent fashion to achieve optimal performance of the whole Client-Server system. As part of Concurrent Requesting, a client is always prepared to make concurrent requests to get the page it needs in the fastest way possible. Concurrent Requesting would require the use of objects such as threads or processes

ing involves dynamically calculating the delays D_p and D_s based how well Peer Caching and Proxy Caching has worked for the client. Please see FIGS. 36 through 38.

[0753] The following is an algorithmic description using pseudo-code of an illustrative embodiment.

[0754] startOurClient is a function that is invoked initially when the client is started.

```

void startOurClient() {
    Initialize the global variable delay to appropriate value based on a
    predefined policy. When delay is positive, it signifies the amount of time to
    wait after Proxy Caching before Peer Caching is attempted; and when
    delay is negative it signifies the amount of time to wait after Peer Caching
    before Proxy Caching is attempted. As an example:
    delay = 50;
    Start a thread for peer responses (i.e., Peer Caching server) with thread
    function as peerServer;
}

getPage function
The function getPage is called by the client's application to get a page. This
function looks up the local cache and if the page is not found, attempts to get the
page from a peer or proxy/server using the ideas of Concurrent Requesting and
Smart Requesting.
void getPage(PageIdType pageId) {
    if pageId present in the local cache then {
        retrieve it and return it to the caller;
    }
    if (delay > 0) {
        myDelay = delay;
        Call requestProxy(pageId);
    }
    else {
        myDelay = -delay;
        Call requestPeer(pageId);
    }
    Wait for gotPage event to be signaled for a maximum of myDelay
    milliseconds;
    If the page was obtained as indicated by gotPage being signaled {
        Modify delay appropriately i.e., if the page was obtained through
        Proxy Caching increment delay else decrement it;
        Return the page;
    }
    if (delay > 0) {
        Call requestPeer(pageId);
    }
    else {
        Call requestProxy(pageId);
    }
    Wait for the page to come through either methods;
    Depending on how the page came (through Proxy Caching or Peer
    Caching) increment or decrement delay;
    Return the page;
}

```

that would allow one to programmatically implement Concurrent Programming. This document assumes the use of threads to describe a possible and preferred way to implement Concurrent Requesting.

[0752] The idea of Smart Requesting includes using an adaptive algorithm to intelligently stagger or schedule requests so that a client, even while using Concurrent Requesting, would not unnecessarily attempt to get a page through more than one means. An example of this is when a client has consistently gotten its page requests fulfilled through Peer Caching in the past. It would come to depend on Peer Caching for future page requests more than the other possible means. On the other hand, if Peer Caching has not worked for that client for some time, it would schedule a proxy request before a Peer Caching request. Smart Request-

[0755] The function requestProxy sends a page request to the proxy and starts a thread that waits for the page response (or times out). The function proxyResponse is the thread function that waits for the response based on the arguments passed to it.

```

void requestProxy(pageId) {
    Send a page request for pageId to a predefined proxy/server as per the
    proxy/server protocol;
    Start a thread with the thread function proxyResponse that waits for
    the response to the request - the function proxyResponse is passed
    arguments: the socket X where it should wait and pageId.
}

```

-continued

```

void proxyResponse(socket X, pageld) {
    Wait at the socket X for a response with a timeout of time TY;
    If a response was received at socket X {
        Uncompress the packet if necessary;
        Validate the packet and ensure that this is a
        valid response to the request and has the page requested (he.,
        match the page Id);
    }
    else {
        // this is time out: didn't receive any
        // response in time TY
        Set appropriate indicator to indicate time-out;
    }
    Signal an event to signify completion of this thread;
}
requestPeer and peerResponse functions

```

[0756] The function requestPeer is similar to requestProxy except that it sends a page request to peers and starts a thread that waits for the page response (or times out). The function peerResponse is the thread function that waits for the response based on the arguments passed to it.

```

Void requestPeer(pageld) {
    Create a UDP socket X bound to port 3002;
    Compose a packet that consists of:
        a code indicating that this is a request for a page
        Some kind of an identifier that uniquely identifies the page
        wanted such as the URL
        other info such as security information or access validators
    Send this packet as a multicast packet to 239.0.0.1:2001 through
    the socket X created above;
    Create a thread with the thread function peerResponse and pass
    socket X and pageld as arguments to it;
}
Void peerResponse(socket X, pageld) {
    Wait at the socket X for a response with a timeout of time TX;
    If a packet was received at socket X {
        Uncompress the packet if necessary;
        Validate the packet and ensure that this is a
        valid response to the request and has the page requested (i.e.,
        match the pageld);
    }
    else {
        // this is time out: didn't receive any
        // response in time TX
        Set appropriate indicator to indicate time-out;
    }
    Signal an event to signify completion of this thread;
}
peerServer function

```

[0757] The function peerServer described below serves page requests received through Peer Caching as multicast packets. The function below describes how this thread would work:

```

void peerServer() {
    Create a multicast socket M bound to port 2001;
    Have M "join" the IP address 239.0.0.1;
    while (not asked to terminate) {
        Wait at M for a multicast packet;
        If a packet is received then {
            Store the source IP addr in S along with the source port number in B;
            Validate the packet that it is a valid request for a page that can be
            served (with valid security credentials);
        }
    }
}

```

-continued

```

    Look up the page id in the local client cache;
    If the page is found {
        Compose a packet that contains the pageld of the
        page as well as the page contents to send;
        Optionally compress the packet before sending;
        Send this packet to the IP address S at port B;
    }
}
}

```

PIRACY PREVENTION FOR STREAMED APPLICATIONS

[0758] Summary

[0759] The details presented in this section describe new techniques of the invention that have been developed to combat software piracy of applications provided over networks, in situations where an ASP's clients' machines execute the software applications locally. The remote ASP server must make all the files that constitute an application available to any subscribed user, because it cannot predict with complete accuracy which files are needed at what point in time. Nor is there a reliable and secure method by which the server can be aware of certain information local to the client computer that could be useful at stopping piracy. The process may be a rogue process intent on pirating the data, or it may be a secure process run from an executable provided by the ASP.

[0760] Aspects of the Invention

[0761] 1. Client-side fine-grained filtering of file accesses directed at remotely served files, for anti-piracy purposes. Traditional network filesystems permit or deny file access at the server side, not the client side. Here, the server provides blanket access to a given user to all the files that the user may need during the execution of an application, and makes more intelligent decisions about which accesses to permit or deny.

[0762] 2. Filtering of file accesses based on where the code for the process that originated the request is stored. Traditional file systems permit or deny file access usually based on the credentials of a user account or process token, not on where the code for the process resides. Here, a filesystem may want to take into account whether the code for the originating process resides in secure remote location or an insecure local location.

[0763] 3. Identification of crucial portions of served files and filtering file accesses depending on the portion targeted. The smallest level of granularity that traditional file systems can operate on is at the level of files, not at the level of the sections contained in the files (for example, whether or not data from a code section or a resource section is requested).

[0764] 4. Filtering of file accesses based on the surmised purpose of the file access, as determined by examining the program stack or flags associated with the request. Traditional file systems do not attempt to determine why a file access was issued before permitting or denying the access, e.g., whether the purpose is to copy the data or page in the data as code for execution.

[0765] 5. Filtering of file accesses based on the surmised purpose of the file access, as determined by examining a history of previous file accesses by the same process.

Traditional file systems do not keep around histories of which blocks a given requester had previously requested from a file. This history can be useful in seeing if the requests match a pattern that suggests a file copy is occurring as opposed to code execution.

[0766] Benefits of the Anti-Piracy Features of the Present Invention

[0767] This is an enabler technology that allows a programmer to build security into a certain type of application delivery system that would otherwise not be possible. Several companies are developing technology that allows an application to be served remotely, but executed locally. Current filesystems provide no way to protect the files that make up this application from being copied and thus pirated. The above techniques are tools that enable a filesystem to allow just those requests that will let the application run normally and block those that are the result of attempts to pirate the application's code or data. This provides a competitive advantage to those software providers who use this technology, because piracy results in lost revenue and, by preventing this, piracy they can prevent this loss.

[0768] The techniques described herein were developed for the purpose of preventing the piracy of computer software programs that are served from a remote server, but executed on a local client. However, they can be used by any computer software security solution that would benefit from the ability to filter file accesses with more flexibility than currently provided by most filesystems.

[0769] When a filesystem receives a request, it must decide whether or not the request should be granted or denied for security reasons. If the target file is local, the filesystem makes the decision by itself, and if the target file is remote, it must ask the server to handle the request for it. The above techniques are ways in which the filesystem can gather more information about the request than it would ordinarily have. It can then use that information to improve the quality of its decisions. Traditional approaches, such as granting a currently logged-in user access to certain files and directories that are marked with his credentials, are not flexible enough for many situations. As for remote files, the server has only a limited amount of information about the client machine. The filesystem at the client side can make grant/deny decisions based on local information before ever asking the server, in order to provide a more intelligent layer of security.

[0770] For example, it may be desirable to allow the user to execute these files, but not copy them. It may be desirable to grant access to only certain processes run by the user, but not others, because it is judged that some processes to be more secure or well-behaved than others. And it may be desirable to allow the user to access only certain sections of these files and from only certain processes for certain periods of time. The above techniques are tools that are added to a filesystem to give it these abilities.

[0771] Overview of the Anti-Piracy Features of the Present Invention

[0772] With respect to FIG. 40, preventing piracy of remotely served, locally executed applications is shown. This figure illustrates the problem of software piracy in an application delivery system, and how it can be stopped using the techniques described in this section. The client computer 4001 is connected to a server 4009 run by an ASP 4007. The server 4009 provides access to application files 4008, out of

which the application executable is run by the client 4001 locally on his machine. (This is Process #14002). However, the user can attempt to access and copy the application files to local storage 4009 on his machine, and thus be able to run them without authorization or give them to another person. But since all requests directed at the remote files 4006 must first pass through the local network filesystem, this filesystem can be enhanced 4005 to deny all such requests that it thinks are the result of an attempt at piracy.

[0773] Referring to FIG. 41, the filtering of accesses to remote application files, illustrating New Technique #1, as described above is shown. (Note: the client computer represented here and in all subsequent figures is part of the same client-server system as in FIG. 40, but the server/ASP diagram has been omitted to save space.) A user 4102 who has been granted access to remotely served files 4106 representing an application is attempting to access these files. The local enhanced network filesystem 4103 is able to deny access to certain files 4105 and grant access to others 4104, for the purpose of protecting critical parts of the application from piracy.

[0774] With respect to FIG. 42, the filtering of accesses to remote files based on process code location, illustrating New Technique #2, as described above, is shown. Here there are two processes on the client computer. Process #14202 has been run from an executable file 4206 that is part of a remotely served application 4207, and process #24203 has been run from a local executable file 4204. They are both attempting to access a remote data file 4206 that is part of the served application 4207. The local enhanced network filesystem 4205 is denying Process #24203 access and granting Process #14202 access because Process #2's 4203 executable is stored locally, and thus is not secure, while Process #1's 4202 executable is provided by the server 4207, and thus can be vouched for.

[0775] Referring to FIG. 43, the filtering of accesses to remote files based on targeted file section, illustrating New Technique #3, as described above, is shown. Here there is a single local process 4302 that is attempting to read from a remotely served executable file 4307. The enhanced network filesystem 4304 is denying an attempt to read from the code section 4306 of the file 4307 while granting an attempt to read from a non-code section 4305 of the file 4307. This is useful when access to some part of the file must be allowed, but access to other parts should be denied to prevent piracy of the entire file.

[0776] With respect to FIG. 44, the filtering of accesses to remote files based on surmised purpose, illustrating New Technique #4 as described above, is shown. Here, two attempts to read from the code section 4407 of a remote executable file 4408 are being made from a process 4402 that was run from this file 4408. However, one request is denied because it originated 4406 from the process's code 4403 itself, while another is approved because it originated from code in the Virtual Memory Subsystem 4404. This prevents even a rogue remote process from attempting to pirate its own code, while allowing legitimate requests for the code to be completed.

[0777] Referring to FIG. 45, the filtering of accesses to remote files based on past access history, illustrating New Technique #5 as described above, is shown. Here, two processes 4502, 4503 run from a local executable 4504 are attempting to access a remote file 4508. The enhanced network filesystem 4507 keeps around a history of previous

file accesses by these processes 4505, 4506, which it consults to make decisions about permitting/denying further accesses. Process #1's 4502 access attempt is granted, while Process #2's 4503 is denied, because the filesystem 4507 detected a suspicious pattern in Process #2's 4503 previous access history 4506.

[0778] Anti-Piracy Details of the Invention

[0779] Five anti-piracy embodiments are disclosed below that can be used by an ASP-installed network filesystem to combat piracy of remotely served applications. The ASP installs a software component on the client that is able to take advantage of local knowledge, e.g., which process on the client originated a request for data, and permit or deny requests for remote files before sending the requests to the server. That is, a network filesystem is installed on the local user's computer that manages access to these remote files. All input/output requests to these files must pass through this filesystem, and if the filesystem determines that a given request is suspicious in some way, it has the freedom to deny it.

[0780] Anti-Piracy Embodiment #1: Client-side Fine-grained Filtering of File Accesses Directed at Remotely Served Files, for Anti-piracy Purposes

[0781] Referring again to FIG. 41, the approach of the first anti-piracy embodiment is that a software component 4102 executing locally on a client computer 4101 has available to it much more information about the state of this computer than does a server providing access to remote files. Thus, the server can filter access only on a much coarser level than this client component. An ASP can take advantage of this by installing a network filesystem 4103 on the client computer that is designated to handle and forward all requests directed at files located on a given remote server. This filesystem 4103 examines each request, and either grants or denies it depending on whether the request is justifiable from a security perspective. It can use information such as the nature of the originating process, the history of previous access by the process, the section of the targeted file being requested, and so on, in order to make its decision.

[0782] The best way known of implementing this approach is to write a network redirector filesystem component 4103 for the operating system that the ASP's clients' machines will be running. This component will be installed, and will make visible to the system a path that represents the server on which the ASP's application files are stored. The local computer can now begin accessing these files, and the filesystem 4103 will be asked to handle requests for these files. On most operating systems, the filesystem 4103 will register dispatch routines to the system that handle common file operations such as open, read, write and close. When a local process 4102 makes a request of an ASP-served file, the OS calls one of these dispatch routines with the request. In the dispatch routine, the filesystem 4103 examines the request and decides whether to deny it or grant it. If granted, it will forward the request to the remote server and send back the response to the operating system.

[0783] Anti-Piracy Embodiment #2: Filtering of File Accesses Based on Where the Code for the Process that Originated the Request is Stored

[0784] Referring again to FIG. 42, when a filesystem 4205 receives a request for access to a given file, the request always originates from a given process on the computer. By determining where the executable file that the process was

run from is located, the network filesystem 4205 can make a more informed decision about the security risk associated with granting the request. For example, if the executable file 4204 is located on the local computer 4202, then it may contain any code whatsoever, code that may attempt to copy and store the contents of any remote files it can gain access to. The filesystem 4205 can reject requests from these processes as being too risky. However, if the executable file 4206 is being served by the ASP's remote server 4207, then the process can assume to be well-behaved, since it is under the control of the ASP. The filesystem 4205 can grant accesses that come from these processes 4202 in confidence that the security risks are minimal.

[0785] The best way known of implementing this approach is to modify a network filesystem 4205 to determine the identity of the process that originated a relevant open, read, or write request for a remote file. On some OSes a unique process ID is embedded in the request, and on others, a system call can be made to get this ID. Then, this ID must be used to look up the pathname of the executable file from which the process was run. To do this, upon initialization the filesystem 4205 must have registered a callback that is invoked whenever a new process is created. When this callback is invoked, the pathname to the process executable and the new process ID are provided as arguments, data which the filesystem 4205 then stores in a data structure. This data structure is consulted while servicing a file request, in order to match the process ID that originated the request with the process's executable. Then the root of the pathname of that executable is extracted. The root uniquely identifies the storage device or remote server that provides the file. If the root specifies an ASP server that is known to be secure, as opposed to a local storage device that is insecure, then the request can be safely granted.

[0786] Anti-Piracy Embodiment #3: Identification of Crucial Portions of Served Files and Filtering File Access Depending on the Portion Targeted

[0787] Referring again to FIG. 43, a served application usually consists of many files. In order to steal the application, a pirate would have to copy at least those files that store the code for the application's primary executable, and perhaps other files as well. This leads to the conclusion that some files are more important than others, and that some portions of some files are most important of all. Ordinarily, the best solution would be to deny access to the primary executable file and its associated executables in its entirety, but this is not usually possible. In order to initially run the application, the filesystem 4304 must grant unrestricted access to some portions of the primary executable. In order to prevent piracy, the filesystem 4304 can grant access selectively to just those portions that are needed. Additionally, the running application 4302 itself does not usually need to read its own code section, but does need to read other sections for purposes such as resource loading. Therefore, additional security can be introduced by denying access to the code sections 4306 of ASP-served executables 4307 even to those executables themselves.

[0788] To implement this, modify a network filesystem's 4304 open file dispatch routine to detect when a remotely served executable 4307 is being opened. When this is detected, the executable file 4307 is examined to determine the offset and length of its code section 4306, and this information is stored in a data structure. On most OSes,

executable files contain headers from which this information can be easily read. In the read and write dispatch routines, the network filesystem 4304 checks if the request is for a remote executable 4307, and if so, the offset and length of the code section 4306 of this executable 4307 is read from the data structure in which it was previously stored. Then the offset and length of the request are checked to see if they intersect the code section 4306 of this executable 4307. If so, the request can be denied.

[0789] Anti-Piracy Embodiment #4: Filtering of File Accesses Based on the Surmised Purpose of the File Access, as Determined by Examining the Program Stack or Flags Associated with the Request

[0790] Referring again to FIG. 44, the approach of the fourth embodiment is that identical requests from the same process for a remotely served file can be distinguished based on the reason the request was issued. For example, on a computer with a virtual memory subsystem 4404, the VMS's own code will be invoked to page-in code for a process that attempts to execute code in pages that are not currently present. To do this, the VMS 4404 must issue a read request to the filesystem 4405 that handles the process' 4402 executable file 4408. Since this request is not for any ulterior purpose, such as piracy, and is necessary for the application to execute, the request should be granted. If the filesystem 4405 gets the originating process ID for such requests, the process whose code is being paged in will be known. However, this same process ID will also be returned for requests that originate as a result of an attempt by the process itself to read its own code (perhaps for the purpose of piracy). Many applications have loopholes that allow the user to execute a macro, for example, that reads and writes arbitrary files. If the filesystem 4405 simply filters requests based on process IDs, it will mistakenly allow users to pirate remotely served applications, as long as they can send the necessary reads and writes from within the remote application itself.

[0791] However, even if the process IDs are the same for two apparently identical requests, there are ways the filesystem 4405 can distinguish them. There are two ways to do this in a manner relevant to combating anti-piracy. The way to implement the first method is to have the filesystem 4405, upon receiving a read request, check for the presence of the paging I/O flag that is supported by several operating systems. If this flag is not present, then the request did not come from the VMS 4404, but from the process itself 4403, and thus the request is risky and not apparently necessary for the application to run. If the flag is present though, the request almost certainly originated from the VMS 4404 for the purpose of reading in code to allow the process to execute. The request should be allowed.

[0792] Another way to make this same determination is to have the filesystem 4405 examine the program stack upon receiving a read request. In several operating systems, a process will attempt to execute code that resides in a virtual page regardless of whether the page is present or not. If the page is not present, a page fault occurs, and a structure is placed onto the stack that holds information about the processor's current state. Then the VMS 4404 gets control. The VMS 4404 then calls the read routine of the filesystem 4405 that handles the process's executable file to read this code into memory. The filesystem 4405 now reads back-

wards up the stack up to a certain point, searching for the presence of the structure that is placed on the stack as a result of a page fault. If such a structure is found, the execution pointer register stored in the structure is examined. If the pointer is a memory address within the boundary of the virtual memory page that is being paged in, then the filesystem 4405 knows the read request is legitimate.

[0793] Anti-Piracy Embodiment #5: Filtering of File Accesses Based on the Surmised Purpose of the File Access, as Determined by Examining a History of Previous File Accesses by the Same Process

[0794] Referring again to FIG. 45, if one looks at the series of file requests that are typically made as a result of attempting to copy an executable file, as opposed to those made in the course of executing that file, one can see certain patterns.

[0795] The copy pattern is usually a sequence of sequentially ordered read requests, while the execution pattern tends to jump around a lot (as the result of code branches into non-present pages). A filesystem can be enhanced to keep around a history of requests made by specific processes on remotely served files. Then, for every subsequent request to such a file, the history for the originating process can be examined to check for certain patterns. If a file-copy pattern is seen, then the pirate may be attempting to steal the file, and the request should be denied. If an execution type pattern is seen, then the user is simply trying to run the application, and the request should be granted.

[0796] To implement this, a filesystem 4507 tells the operating system, via an operating system call, upon initialization, to call it back whenever a new process is created. When it is called back, the filesystem 4507 creates a new data structure for the process that will store file access histories 4505, 4506. Then, in its read-file dispatch routines, the filesystem 4507 determines the process ID of the originating process, and examines the process's access history 4505, 4506. It only examines entries in that history 4505, 4506 that refer to the file currently being requested. It will then run a heuristic algorithm that tries to determine if the pattern of accesses more closely resembles an attempted file copy than code execution. An effective algorithm is to simply see if the past *n* read requests to this file have been sequential, where *n* is some constant. If so, then the request is denied. If not, then the request is granted. In either case, an entry is made to the filesystem's process access history 4505, 4506 that records the file name, offset, and length of the request made by that process to this file.

CONCLUSION

[0797] Although the present invention has been described using particular illustrative embodiments, it will be understood that many variations in construction, arrangement and use are possible within the scope of this invention. Other embodiments may use different network protocols, different programming techniques, or different heuristics, in each component block of the invention. Specific examples of variations include:

[0798] The proxy used in Proxy Caching could be anywhere in the Internet along the network path between a Client and the Server; and

[0799] Concurrent Requesting and Smart Requesting can be implemented in hardware instead of software.

[0800] A number of insubstantial variations are possible in the implementation of anti-piracy features of the invention. For example, instead of modifying the filesystem proper to provide anti-piracy features, a network proxy component can be placed on the client computer to filter network requests made by a conventional local network filesystem. These requests generally correspond to requests for remote files made to the filesystem by a local process, and the type of filtering taught by the present invention can be performed on these requests. A filesystem filter component can also be written to implement these methods, instead of modifying the filesystem itself.

[0801] Although the invention is described herein with reference to the preferred embodiment, one skilled in the art will readily appreciate that other applications may be substituted for those set forth herein without departing from the spirit and scope of the present invention. Accordingly, the invention should only be limited by the Claims included below.

1. A process for client-side retrieval, storage, and execution of application programs and other related data streamed from a server across a computer network to a client system in a computer environment, comprising the steps of:

providing a streaming file system on said client;

wherein said streaming file system appears to said client to contain the installed application program;

wherein said streaming file system receives all requests from local processes for application program code or data that are part of the application;

providing a persistent cache on said client;

wherein said streaming file system satisfies requests for application program code or data by retrieving it from said persistent cache stored in a native file system or by retrieving it directly from said server; and

wherein application program code or data retrieved from said server is placed in said persistent cache for reuse.

2. The process of claim 1, wherein said persistent cache is encrypted with a key not permanently stored on said client to prevent unauthorized use or duplication of application code or data; and wherein said key is sent to said client upon application startup from said server and said key is not stored in the application program's persistent storage area in said persistent cache.

3. The process of claim 1, wherein said client initiates the prefetching of application program code and data from said server; and wherein said client inspects program code or data file requests and consults the contents of said persistent cache as well as historic information about application program fetching patterns and uses this information to request additional blocks of application program code and data from said server that said client expects will be needed soon.

4. The process of claim 1, wherein said server initiates the prefetching of application program code and data for said client; and wherein said server examines the patterns of requests made by said client and selectively returns to said client additional blocks that said client did not request but is likely to need soon.

5. The process of claim 1, further comprising the step of: providing a client-to-client communication mechanism that allows local application customization to migrate from one client machine to another without involving server communication;

wherein when a user wishes to run an application on a second machine, but wishes to retain customizations made previously on the first, said client-to-client mechanism contacts the first machine to retrieve customized files and other customization data; and

wherein unmodified files are retrieved from said server.

6. The process of claim 1, further comprising the step of:

wherein said streaming file system is a copy-on-write file system that allows applications to write configuration or initialization files where they want to without rewriting the application, and without disturbing the local customization of other clients;

wherein each block of data in said persistent cache is marked as clean or dirty;

wherein pages marked as dirty have been customized by the application program and cannot be removed from the cache without losing client customization;

wherein pages marked as clean may be purged from the cache because they can be retrieved again from said server;

providing a cache index;

wherein said cache index indicates which pages in said persistent cache are clean and dirty; and

wherein said copy-on-write file system references said cache index to determine if a page is clean or dirty.

7. The process of claim 1, further comprising the step of: marking specific files in said persistent cache as not modifiable;

wherein said streaming file system does not allow any data to be written to said specific files that are marked as not modifiable; and

wherein attempts by any processes to mark any of said specific files as modifiable will not succeed.

8. The process of claim 1, further comprising the step of:

maintaining checksums of application code and data in said persistent cache;

wherein when a block of code or data is requested by a local process said streaming file system computes the checksum of the data block before it is returned to the local process; and

wherein if a computed checksum does not match the checksum stored in said persistent cache the cache entry is invalidated and a fresh copy of the page is retrieved from said server.

9. The process of claim 1, further comprising the step of:

assigning each file in an application program a unique identifier;

wherein files that are changed or added in an application upgrade are given new identifiers never before used for that application program;

wherein files that are unchanged retain the same number; and

wherein directories whose contents change are also considered changes. If any file changes, this will cause its parent to change, all the way up to the root directory.

10. The process of claim 9, wherein when an application upgrade occurs said client is given a new root directory for the application program by said server; wherein said new root directory is used by said streaming file system to search for files in the application program; wherein files that do not change can be reused from said persistent cache without downloading them again from said server; and wherein files with new identifiers are retrieved from said server.

11. The process of claim 9, wherein said application upgrades can be marked as mandatory by said server causing the new root directory for the application program to be used immediately; and wherein said streaming file system contacts said server when an application program is started in order to receive any application upgrades.

12. The process of claim 1, wherein said server broadcasts an application program's code and data and any client that is interested in that particular application program stores the broadcasted code and data for later use.

13. A process for client-side retrieval, storage, and execution of application programs and other related data streamed from a server across a computer network to a client system in a computer environment, comprising the steps of:

providing a kernel-mode streaming file system driver on said client;

providing a user-mode client on said client;

wherein said streaming file system receives all requests from local processes for application program code or data that are part of the application;

providing a persistent cache on said client;

wherein requests made to said streaming file system are directed to said user-mode client or retrieved from said persistent cache;

wherein said user-mode client handles the application program code and data streams from said server and sends the results back to said streaming file system driver; and

wherein application program code or data retrieved from said server is placed in said persistent cache for reuse.

14. The process of claim 13, wherein said persistent cache is encrypted with a key not permanently stored on said client to prevent unauthorized use or duplication of application code or data; and wherein said key is sent to said client upon application startup from said server and said key is not stored in the application program's persistent storage area in said persistent cache.

15. The process of claim 13, wherein said client initiates the prefetching of application program code and data from said server; and wherein said client inspects program code or data file requests and consults the contents of said persistent cache as well as historic information about application program fetching patterns and uses this information to request additional blocks of application program code and data from said server that said client expects will be needed soon.

16. The process of claim 13, wherein said server initiates the prefetching of application program code and data for said client; and wherein said server examines the patterns of requests made by said client and selectively returns to said client additional blocks that said client did not request but is likely to need soon.

17. The process of claim 13, further comprising the step of:

providing a client-to-client communication mechanism that allows local application customization to migrate from one client machine to another without involving server communication;

wherein when a user wishes to run an application on a second machine, but wishes to retain customizations made previously on the first, said client-to-client mechanism contacts the first machine to retrieve customized files and other customization data; and

wherein unmodified files are retrieved from said server.

18. The process of claim 13, further comprising the step of:

wherein said streaming file system is a copy-on-write file system that allows applications to write configuration or initialization files where they want to without rewriting the application, and without disturbing the local customization of other clients;

wherein each block of data in said persistent cache is marked as clean or dirty;

wherein pages marked as dirty have been customized by the application program and cannot be removed from the cache without losing client customization;

wherein pages marked as clean may be purged from the cache because they can be retrieved again from said server;

providing a cache index;

wherein said cache index indicates which pages in said persistent cache are clean and dirty; and

wherein said copy-on-write file system references said cache index to determine if a page is clean or dirty.

19. The process of claim 13, further comprising the step of:

marking specific files in said persistent cache as not modifiable;

wherein said streaming file system does not allow any data to be written to said specific files that are marked as not modifiable; and

wherein attempts by any processes to mark any of said specific files as modifiable will not succeed.

20. The process of claim 13, further comprising the step of:

maintaining checksums of application code and data in said persistent cache;

wherein when a block of code or data is requested by a local process said streaming file system computes the checksum of the data block before it is returned to the local process; and

wherein if a computed checksum does not match the checksum stored in said persistent cache the cache entry is invalidated and a fresh copy of the page is retrieved from said server.

21. The process of claim 13, further comprising the step of:

assigning each file in an application program a unique identifier;

wherein files that are changed or added in an application upgrade are given new identifiers never before used for that application program;

wherein files that are unchanged retain the same number; and

wherein directories whose contents change are also considered changes. If any file changes, this will cause its parent to change, all the way up to the root directory.

22. The process of claim 21, wherein when an application upgrade occurs said client is given a new root directory for the application program by said server; wherein said new root directory is used by said streaming file system to search for files in the application program; wherein files that do not change can be reused from said persistent cache without downloading them again from said server; and wherein files with new identifiers are retrieved from said server.

23. The process of claim 21, wherein said application upgrades can be marked as mandatory by said server causing the new root directory for the application program to be used immediately; and wherein said streaming file system contacts said server when an application program is started in order to receive any application upgrades.

24. The process of claim 13, wherein said server broadcasts an application program's code and data and any client that is interested in that particular application program stores the broadcasted code and data for later use.

25. A process for client-side retrieval, storage, and execution of application programs and other related data streamed from a server across a computer network to a client system in a computer environment, comprising the steps of:

providing a streaming block driver on said client;

wherein said block driver provides the abstraction of a physical disk to a native file system already installed on the client operating system;

providing a persistent cache on said client;

wherein said block driver receives requests for physical block reads and writes from local processes which it satisfies out of said persistent cache on a standard file system that is backed by a physical disk drive; and

wherein requests that cannot be satisfied by said persistent cache are sent to said server.

26. The process of claim 25, wherein said persistent cache is encrypted with a key not permanently stored on said client to prevent unauthorized use or duplication of application code or data; and wherein said key is sent to said client upon application startup from said server and said key is not stored in the application program's persistent storage area in said persistent cache.

27. The process of claim 25, wherein said client initiates the prefetching of application program code and data from said server; and wherein said client inspects program code or data file requests and consults the contents of said persistent

cache as well as historic information about application program fetching patterns and uses this information to request additional blocks of application program code and data from said server that said client expects will be needed soon.

28. The process of claim 25, wherein said server initiates the prefetching of application program code and data for said client; and wherein said server examines the patterns of requests made by said client and selectively returns to said client additional blocks that said client did not request but is likely to need soon.

29. The process of claim 25, further comprising the step of:

providing a client-to-client communication mechanism that allows local application customization to migrate from one client machine to another without involving server communication;

wherein when a user wishes to run an application on a second machine, but wishes to retain customizations made previously on the first, said client-to-client mechanism contacts the first machine to retrieve customized files and other customization data; and

wherein unmodified files are retrieved from said server.

30. The process of claim 25, further comprising the step of:

wherein said block driver is a copy-on-write block driver that allows applications to write configuration or initialization files where they want to without rewriting the application, and without disturbing the local customization of other clients;

wherein each block of data in said persistent cache is marked as clean or dirty;

wherein pages marked as dirty have been customized by the application program and cannot be removed from the cache without losing client customization;

wherein pages marked as clean may be purged from the cache because they can be retrieved again from said server;

providing a cache index;

wherein said cache index indicates which pages in said persistent cache are clean and dirty; and

wherein said copy-on-write block driver references said cache index to determine if a page is clean or dirty.

31. The process of claim 25, further comprising the step of:

marking specific files in said persistent cache as not modifiable;

wherein said block driver does not allow any data to be written to said specific files that are marked as not modifiable; and

wherein attempts by any processes to mark any of said specific files as modifiable will not succeed.

32. The process of claim 25, further comprising the step of:

maintaining checksums of application code and data in said persistent cache;

wherein when a block of code or data is requested by a local process said streaming file system computes the checksum of the data block before it is returned to the local process; and

wherein if a computed checksum does not match the checksum stored in said persistent cache the cache entry is invalidated and a fresh copy of the page is retrieved from said server.

33. The process of claim 25, further comprising the step of:

assigning each file in an application program a unique identifier;

wherein files that are changed or added in an application upgrade are given new identifiers never before used for that application program; wherein files that are unchanged retain the same number; and

wherein directories whose contents change are also considered changes.

If any file changes, this will cause its parent to change, all the way up to the root directory.

34. The process of claim 33, wherein when an application upgrade occurs said client is given a new root directory for the application program by said server; wherein said new root directory is used by said block driver to search for files in the application program; wherein files that do not change can be reused from said persistent cache without downloading them again from said server; and wherein files with new identifiers are retrieved from said server.

35. The process of claim 33, wherein said application upgrades can be marked as mandatory by said server causing the new root directory for the application program to be used immediately; and wherein said block driver contacts said server when an application program is started in order to receive any application upgrades.

36. The process of claim 25, wherein said server broadcasts an application program's code and data and any client that is interested in that particular application program stores the broadcasted code and data for later use.

37. A process for client-side retrieval, storage, and execution of application programs and other related data streamed from a server across a computer network to a client system in a computer environment, comprising the steps of:

providing a disk driver on said client;

providing a user mode client on said client;

wherein said disk driver sends all file requests that it receives to said user-mode client;

providing a persistent cache on said client; and

wherein said user-mode client attempts to satisfy said file requests from said program cache or by making requests from said server.

38. The process of claim 37, wherein said persistent cache is encrypted with a key not permanently stored on said client to prevent unauthorized use or duplication of application code or data; and wherein said key is sent to said client upon application startup from said server and said key is not stored in the application program's persistent storage area in said persistent cache.

39. The process of claim 37, wherein said client initiates the prefetching of application program code and data from said server; and wherein said client inspects program code or data file requests and consults the contents of said persistent cache as well as historic information about application program fetching patterns and uses this information to request additional blocks of application program code and data from said server that said client expects will be needed soon.

40. The process of claim 37, wherein said server initiates the prefetching of application program code and data for said client; and wherein said server examines the patterns of requests made by said client and selectively returns to said client additional blocks that said client did not request but is likely to need soon.

41. The process of claim 37, further comprising the step of:

providing a client-to-client communication mechanism that allows local application customization to migrate from one client machine to another without involving server communication;

wherein when a user wishes to run an application on a second machine, but wishes to retain customizations made previously on the first, said client-to-client mechanism contacts the first machine to retrieve customized files and other customization data; and

wherein unmodified files are retrieved from said server.

42. The process of claim 37, further comprising the step of:

wherein said user-mode client is a copy-on-write user-mode client that allows applications to write configuration or initialization files where they want to without rewriting the application, and without disturbing the local customization of other clients;

wherein each block of data in said persistent cache is marked as clean or dirty;

wherein pages marked as dirty have been customized by the application program and cannot be removed from the cache without losing client customization;

wherein pages marked as clean may be purged from the cache because they can be retrieved again from said server;

providing a cache index;

wherein said cache index indicates which pages in said persistent cache are clean and dirty; and

wherein said copy-on-write user-mode client references said cache index to determine if a page is clean or dirty.

43. The process of claim 37, further comprising the step of:

marking specific files in said persistent cache as not modifiable;

wherein said user-mode client does not allow any data to be written to said specific files that are marked as not modifiable; and

wherein attempts by any processes to mark any of said specific files as modifiable will not succeed.

44. The process of claim 37, further comprising the step of:

maintaining checksums of application code and data in said persistent cache;

wherein when a block of code or data is requested by a local process said streaming file system computes the checksum of the data block before it is returned to the local process; and

wherein if a computed checksum does not match the checksum stored in said persistent cache the cache entry is invalidated and a fresh copy of the page is retrieved from said server.

45. The process of claim 37, further comprising the step of:

assigning each file in an application program a unique identifier;

wherein files that are changed or added in an application upgrade are given new identifiers never before used for that application program;

wherein files that are unchanged retain the same number; and

wherein directories whose contents change are also considered changes. If any file changes, this will cause its parent to change, all the way up to the root directory.

46. The process of claim 45, wherein when an application upgrade occurs said client is given a new root directory for the application program by said server; wherein said new root directory is used by said user-mode client to search for files in the application program; wherein files that do not change can be reused from said persistent cache without downloading them again from said server; and wherein files with new identifiers are retrieved from said server.

47. The process of claim 45, wherein said application upgrades can be marked as mandatory by said server causing the new root directory for the application program to be used immediately; and wherein said user-mode client contacts said server when an application program is started in order to receive any application upgrades.

48. The process of claim 37, wherein said server broadcasts an application program's code and data and any client that is interested in that particular application program stores the broadcasted code and data for later use.

49. An apparatus for client-side retrieval, storage, and execution of application programs and other related data streamed from a server across a computer network to a client system in a computer environment, comprising:

a streaming file system on said client;

wherein said streaming file system appears to said client to contain the installed application program;

wherein said streaming file system receives all requests from local processes for application program code or data that are part of the application;

a persistent cache on said client;

wherein said streaming file system satisfies requests for application program code or data by retrieving it from said persistent cache stored in a native file system or by retrieving it directly from said server; and

wherein application program code or data retrieved from said server is placed in said persistent cache for reuse.

50. The apparatus of claim 49, wherein said persistent cache is encrypted with a key not permanently stored on said client to prevent unauthorized use or duplication of application code or data; and wherein said key is sent to said client upon application startup from said server and said key is not stored in the application program's persistent storage area in said persistent cache.

51. The apparatus of claim 49, wherein said client initiates the prefetching of application program code and data from said server; and wherein said client inspects program code or data file requests and consults the contents of said persistent cache as well as historic information about application program fetching patterns and uses this information to request additional blocks of application program code and data from said server that said client expects will be needed soon.

52. The apparatus of claim 49, wherein said server initiates the prefetching of application program code and data for said client; and wherein said server examines the patterns of requests made by said client and selectively returns to said client additional blocks that said client did not request but is likely to need soon.

53. The apparatus of claim 49, further comprising:

a client-to-client communication mechanism that allows local application customization to migrate from one client machine to another without involving server communication;

wherein when a user wishes to run an application on a second machine, but wishes to retain customizations made previously on the first, said client-to-client mechanism contacts the first machine to retrieve customized files and other customization data; and

wherein unmodified files are retrieved from said server.

54. The apparatus of claim 49, further comprising:

wherein said streaming file system is a copy-on-write file system that allows applications to write configuration or initialization files where they want to without rewriting the application, and without disturbing the local customization of other clients;

wherein each block of data in said persistent cache is marked as clean or dirty;

wherein pages marked as dirty have been customized by the application program and cannot be removed from the cache without losing client customization;

wherein pages marked as clean may be purged from the cache because they can be retrieved again from said server;

a cache index;

wherein said cache index indicates which pages in said persistent cache are clean and dirty; and

wherein said copy-on-write file system references said cache index to determine if a page is clean or dirty.

55. The apparatus of claim 49, further comprising:

a module for marking specific files in said persistent cache as not modifiable;

wherein said streaming file system does not allow any data to be written to said specific files that are marked as not modifiable; and

wherein attempts by any processes to mark any of said specific files as modifiable will not succeed.

56. The apparatus of claim 49, further comprising:

a module for maintaining checksums of application code and data in said persistent cache;

wherein when a block of code or data is requested by a local process said streaming file system computes the checksum of the data block before it is returned to the local process; and

wherein if a computed checksum does not match the checksum stored in said persistent cache the cache entry is invalidated and a fresh copy of the page is retrieved from said server.

57. The apparatus of claim 49, further comprising:

a module for assigning each file in an application program a unique identifier;

wherein files that are changed or added in an application upgrade are given new identifiers never before used for that application program;

wherein files that are unchanged retain the same number; and

wherein directories whose contents change are also considered changes. If any file changes, this will cause its parent to change, all the way up to the root directory.

58. The apparatus of claim 57, wherein when an application upgrade occurs said client is given a new root directory for the application program by said server; wherein said new root directory is used by said streaming file system to search for files in the application program; wherein files that do not change can be reused from said persistent cache without downloading them again from said server; and wherein files with new identifiers are retrieved from said server.

59. The apparatus of claim 57, wherein said application upgrades can be marked as mandatory by said server causing the new root directory for the application program to be used immediately; and wherein said streaming file system contacts said server when an application program is started in order to receive any application upgrades.

60. The apparatus of claim 49, wherein said server broadcasts an application program's code and data and any client that is interested in that particular application program stores the broadcasted code and data for later use.

61. An apparatus for client-side retrieval, storage, and execution of application programs and other related data streamed from a server across a computer network to a client system in a computer environment, comprising:

a kernel-mode streaming file system driver on said client;

a user-mode client on said client;

wherein said streaming file system receives all requests from local processes for application program code or data that are part of the application;

a persistent cache on said client;

wherein requests made to said streaming file system are directed to said user-mode client or retrieved from said persistent cache;

wherein said user-mode client handles the application program code and data streams from said server and sends the results back to said streaming file system driver; and

wherein application program code or data retrieved from said server is placed in said persistent cache for reuse.

62. The apparatus of claim 61, wherein said persistent cache is encrypted with a key not permanently stored on said client to prevent unauthorized use or duplication of application code or data; and wherein said key is sent to said client upon application startup from said server and said key is not stored in the application program's persistent storage area in said persistent cache.

63. The apparatus of claim 61, wherein said client initiates the prefetching of application program code and data from said server; and wherein said client inspects program code or data file requests and consults the contents of said persistent cache as well as historic information about application program fetching patterns and uses this information to request additional blocks of application program code and data from said server that said client expects will be needed soon.

64. The apparatus of claim 61, wherein said server initiates the prefetching of application program code and data for said client; and wherein said server examines the patterns of requests made by said client and selectively returns to said client additional blocks that said client did not request but is likely to need soon.

65. The apparatus of claim 61, further comprising:

a client-to-client communication mechanism that allows local application customization to migrate from one client machine to another without involving server communication;

wherein when a user wishes to run an application on a second machine, but wishes to retain customizations made previously on the first, said client-to-client mechanism contacts the first machine to retrieve customized files and other customization data; and

wherein unmodified files are retrieved from said server.

66. The apparatus of claim 61, further comprising:

wherein said streaming file system is a copy-on-write file system that allows applications to write configuration or initialization files where they want to without rewriting the application, and without disturbing the local customization of other clients;

wherein each block of data in said persistent cache is marked as clean or dirty;

wherein pages marked as dirty have been customized by the application program and cannot be removed from the cache without losing client customization;

wherein pages marked as clean may be purged from the cache because they can be retrieved again from said server;

a cache index;

wherein said cache index indicates which pages in said persistent cache are clean and dirty; and

wherein said copy-on-write file system references said cache index to determine if a page is clean or dirty.

67. The apparatus of claim 61, further comprising:

a module for marking specific files in said persistent cache as not modifiable;

wherein said streaming file system does not allow any data to be written to said specific files that are marked as not modifiable; and

wherein attempts by any processes to mark any of said specific files as modifiable will not succeed.

68. The apparatus of claim 61, further comprising:

a module for maintaining checksums of application code and data in said persistent cache;

wherein when a block of code or data is requested by a local process said streaming file system computes the checksum of the data block before it is returned to the local process; and

wherein if a computed checksum does not match the checksum stored in said persistent cache the cache entry is invalidated and a fresh copy of the page is retrieved from said server.

69. The apparatus of claim 61, further comprising:

a module for assigning each file in an application program a unique identifier;

wherein files that are changed or added in an application upgrade are given new identifiers never before used for that application program;

wherein files that are unchanged retain the same number; and

wherein directories whose contents change are also considered changes. If any file changes, this will cause its parent to change, all the way up to the root directory.

70. The apparatus of claim 69, wherein when an application upgrade occurs said client is given a new root directory for the application program by said server; wherein said new root directory is used by said streaming file system to search for files in the application program; wherein files that do not change can be reused from said persistent cache without downloading them again from said server; and wherein files with new identifiers are retrieved from said server.

71. The apparatus of claim 69, wherein said application upgrades can be marked as mandatory by said server causing the new root directory for the application program to be used immediately; and wherein said streaming file system contacts said server when an application program is started in order to receive any application upgrades.

72. The apparatus of claim 61, wherein said server broadcasts an application program's code and data and any client that is interested in that particular application program stores the broadcasted code and data for later use.

73. An apparatus for client-side retrieval, storage, and execution of application programs and other related data streamed from a server across a computer network to a client system in a computer environment, comprising:

a streaming block driver on said client;

wherein said block driver provides the abstraction of a physical disk to a native file system already installed on the client operating system;

a persistent cache on said client;

wherein said block driver receives requests for physical block reads and writes from local processes which it satisfies out of said persistent cache on a standard file system that is backed by a physical disk drive; and

wherein requests that cannot be satisfied by said persistent cache are sent to said server.

74. The apparatus of claim 73, wherein said persistent cache is encrypted with a key not permanently stored on said client to prevent unauthorized use or duplication of application code or data; and wherein said key is sent to said client upon application startup from said server and said key is not stored in the application program's persistent storage area in said persistent cache.

75. The apparatus of claim 73, wherein said client initiates the prefetching of application program code and data from said server; and wherein said client inspects program code or data file requests and consults the contents of said persistent cache as well as historic information about application program fetching patterns and uses this information to request additional blocks of application program code and data from said server that said client expects will be needed soon.

76. The apparatus of claim 73, wherein said server initiates the prefetching of application program code and data for said client; and wherein said server examines the patterns of requests made by said client and selectively returns to said client additional blocks that said client did not request but is likely to need soon.

77. The apparatus of claim 73, further comprising:

a client-to-client communication mechanism that allows local application customization to migrate from one client machine to another without involving server communication;

wherein when a user wishes to run an application on a second machine, but wishes to retain customizations made previously on the first, said client-to-client mechanism contacts the first machine to retrieve customized files and other customization data; and

wherein unmodified files are retrieved from said server.

78. The apparatus of claim 73, further comprising:

wherein said block driver is a copy-on-write block driver that allows applications to write configuration or initialization files where they want to without rewriting the application, and without disturbing the local customization of other clients;

wherein each block of data in said persistent cache is marked as clean or dirty;

wherein pages marked as dirty have been customized by the application program and cannot be removed from the cache without losing client customization;

wherein pages marked as clean may be purged from the cache because they can be retrieved again from said server;

a cache index;

wherein said cache index indicates which pages in said persistent cache are clean and dirty; and

wherein said copy-on-write block driver references said cache index to determine if a page is clean or dirty.

79. The apparatus of claim 73, further comprising:

a module for marking specific files in said persistent cache as not modifiable;

wherein said block driver does not allow any data to be written to said specific files that are marked as not modifiable; and

wherein attempts by any processes to mark any of said specific files as modifiable will not succeed.

80. The apparatus of claim 73, further comprising:
 a module for maintaining checksums of application code and data in said persistent cache;
 wherein when a block of code or data is requested by a local process said streaming file system computes the checksum of the data block before it is returned to the local process; and
 wherein if a computed checksum does not match the checksum stored in said persistent cache the cache entry is invalidated and a fresh copy of the page is retrieved from said server.

81. The apparatus of claim 73, further comprising:
 a module for assigning each file in an application program a unique identifier;
 wherein files that are changed or added in an application upgrade are given new identifiers never before used for that application program;
 wherein files that are unchanged retain the same number; and
 wherein directories whose contents change are also considered changes. If any file changes, this will cause its parent to change, all the way up to the root directory.

82. The apparatus of claim 81, wherein when an application upgrade occurs said client is given a new root directory for the application program by said server; wherein said new root directory is used by said block driver to search for files in the application program; wherein files that do not change can be reused from said persistent cache without downloading them again from said server; and wherein files with new identifiers are retrieved from said server.

83. The apparatus of claim 81, wherein said application upgrade can be marked as mandatory by said server causing the new root directory for the application program to be used immediately; and wherein said block driver contacts said server when an application program is started in order to receive any application upgrades.

84. The apparatus of claim 73, wherein said server broadcasts an application program's code and data and any client that is interested in that particular application program stores the broadcasted code and data for later use.

85. An apparatus for client-side retrieval, storage, and execution of application programs and other related data streamed from a server across a computer network to a client system in a computer environment, comprising:
 a disk driver on said client;
 a user mode client on said client;
 wherein said disk driver sends all file requests that it receives to said user-mode client;
 a persistent cache on said client; and
 wherein said user-mode client attempts to satisfy said file requests from said program cache or by making requests from said server.

86. The apparatus of claim 85, wherein said persistent cache is encrypted with a key not permanently stored on said client to prevent unauthorized use or duplication of application code or data; and wherein said key is sent to said

client upon application startup from said server and said key is not stored in the application program's persistent storage area in said persistent cache.

87. The apparatus of claim 85, wherein said client initiates the prefetching of application program code and data from said server; and wherein said client inspects program code or data file requests and consults the contents of said persistent cache as well as historic information about application program fetching patterns and uses this information to request additional blocks of application program code and data from said server that said client expects will be needed soon.

88. The apparatus of claim 85, wherein said server initiates the prefetching of application program code and data for said client; and wherein said server examines the patterns of requests made by said client and selectively returns to said client additional blocks that said client did not request but is likely to need soon.

89. The apparatus of claim 85, further comprising:

a client-to-client communication mechanism that allows local application customization to migrate from one client machine to another without involving server communication;

wherein when a user wishes to run an application on a second machine, but wishes to retain customizations made previously on the first, said client-to-client mechanism contacts the first machine to retrieve customized files and other customization data; and

wherein unmodified files are retrieved from said server.

90. The apparatus of claim 85, further comprising:

wherein said user-mode client is a copy-on-write user-mode client that allows applications to write configuration or initialization files where they want to without rewriting the application, and without disturbing the local customization of other clients;

wherein each block of data in said persistent cache is marked as clean or dirty;

wherein pages marked as dirty have been customized by the application program and cannot be removed from the cache without losing client customization;

wherein pages marked as clean may be purged from the cache because they can be retrieved again from said server;

a cache index;

wherein said cache index indicates which pages in said persistent cache are clean and dirty; and

wherein said copy-on-write user-mode client references said cache index to determine if a page is clean or dirty.

91. The apparatus of claim 85, further comprising:

a module for marking specific files in said persistent cache as not modifiable;

wherein said user-mode client does not allow any data to be written to said specific files that are marked as not modifiable; and

wherein attempts by any processes to mark any of said specific files as modifiable will not succeed.

92. The apparatus of claim 85, further comprising:

a module for maintaining checksums of application code and data in said persistent cache;

wherein when a block of code or data is requested by a local process said streaming file system computes the checksum of the data block before it is returned to the local process; and

wherein if a computed checksum does not match the checksum stored in said persistent cache the cache entry is invalidated and a fresh copy of the page is retrieved from said server.

93. The apparatus of claim 85, further comprising:

a module for assigning each file in an application program a unique identifier;

wherein files that are changed or added in an application upgrade are given new identifiers never before used for that application program;

wherein files that are unchanged retain the same number; and

wherein directories whose contents change are also considered changes. If any file changes, this will cause its parent to change, all the way up to the root directory.

94. The apparatus of claim 93, wherein when an application upgrade occurs said client is given a new root directory for the application program by said server; wherein said new root directory is used by said user-mode client to search for files in the application program; wherein files that do not change can be reused from said persistent cache without downloading them again from said server; and wherein files with new identifiers are retrieved from said server.

95. The apparatus of claim 93, wherein said application upgrades can be marked as mandatory by said server causing the new root directory for the application program to be used immediately; and wherein said user-mode client contacts said server when an application program is started in order to receive any application upgrades.

96. The apparatus of claim 85, wherein said server broadcasts an application program's code and data and any client that is interested in that particular application program stores the broadcasted code and data for later use.

* * * * *



US006412009B1

(12) **United States Patent**
Erickson et al.

(10) **Patent No.:** **US 6,412,009 B1**
(45) **Date of Patent:** **Jun. 25, 2002**

(54) **METHOD AND SYSTEM FOR PROVIDING A PERSISTENT HTTP TUNNEL**

(75) Inventors: **Rodger D. Erickson**, St. Louis, MO (US); **Ronald D. Sanders**, Spokane, WA (US)

(73) Assignee: **Wall Data Incorporated**, Kirkland, WA (US)

(*) Notice: Subject to any disclaimer, the term of this patent is extended or adjusted under 35 U.S.C. 154(b) by 0 days.

(21) Appl. No.: **09/268,068**

(22) Filed: **Mar. 15, 1999**

(51) Int. Cl.⁷ **G06F 13/00**

(52) U.S. Cl. **709/228; 709/217; 709/236; 709/313**

(58) Field of Search **709/203, 217, 709/219, 223, 224, 225, 227, 228, 236, 238, 313**

(56) **References Cited**

U.S. PATENT DOCUMENTS

5,754,830 A * 5/1998 Butts et al. 395/500.44
5,778,372 A * 7/1998 Cordell et al. 707/100
5,935,212 A * 8/1999 Kalajan et al. 709/228
5,941,988 A * 8/1999 Bhagwat et al. 713/201

OTHER PUBLICATIONS

Provan, D., "Tunneling IPX Traffic through IP Networks," RFC 1234, Novell, Inc. (Jun. 1991).
Woodburn, R. et al., "A Scheme for an Internet Encapsulation Protocol: Version 1," RFC 1241, University of Delaware (Jul. 1991).

* cited by examiner

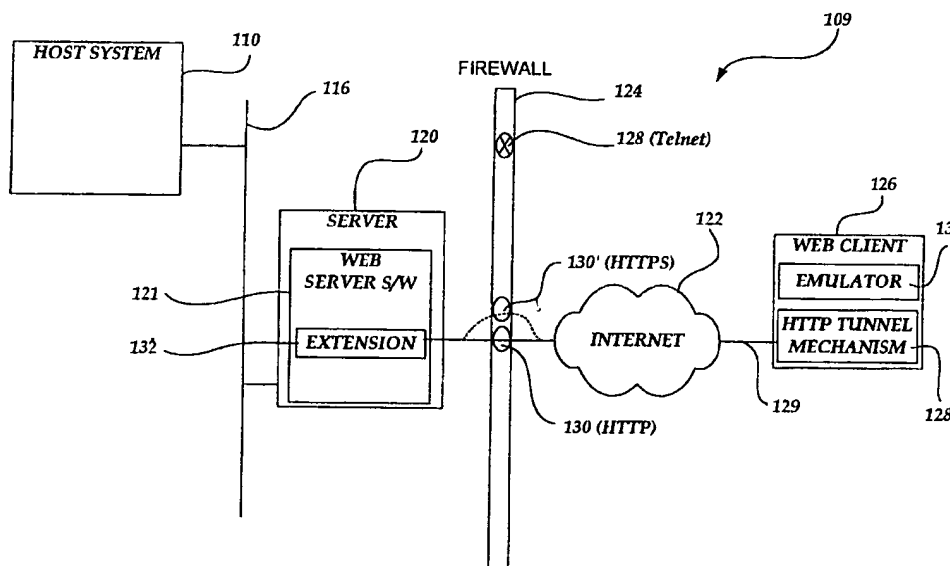
Primary Examiner—Viet D. Vu

(74) *Attorney, Agent, or Firm*—Christensen O'Connor Johnson Kindness PLLC

(57) **ABSTRACT**

A method and system for providing a persistent HTTP tunnel for a connection-oriented protocol between a client and a Web server. A data message complying with the connection-oriented protocol is generated and embedded into a chunked data message in compliance with a chunking option for the HTTP. The chunked data message is transmitted between a client and a Web server. Upon receiving any chunked data message at the Web server, the Web server parses the chunked data message and delivers the data message to a host system. Upon receiving any chunked data message at the client, the client parses the chunked data message and delivers the data message to a terminal emulator running on the client. This allows a terminal session to be supported by a real-time bi-directional persistent connection with the host system. The bi-directional persistent connection allows interleaving of the chunked data messages from the Web client with the chunked data messages on the Web server on the persistent HTTP tunnel.

8 Claims, 5 Drawing Sheets



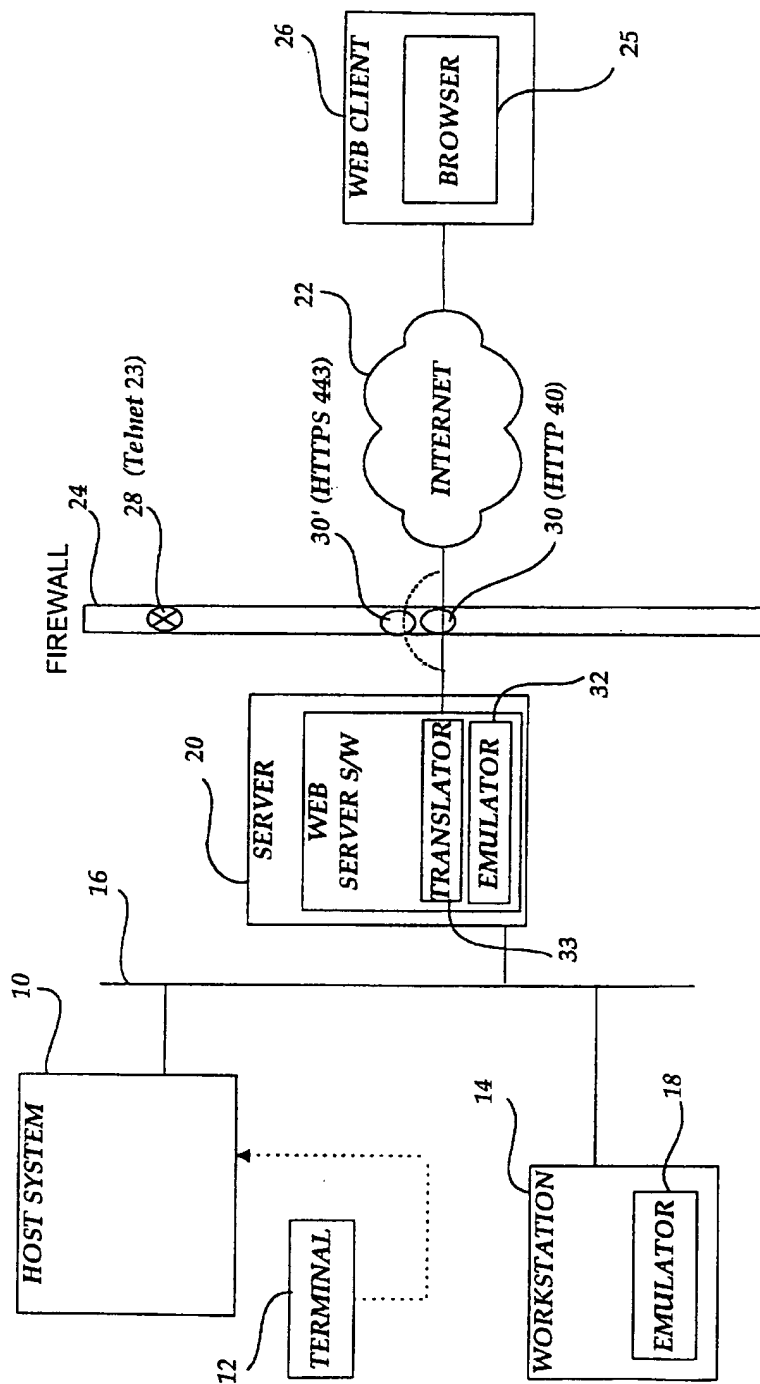
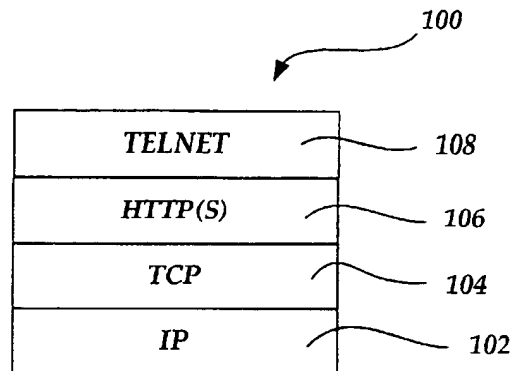
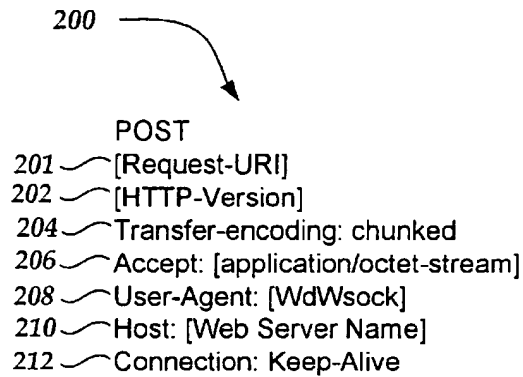
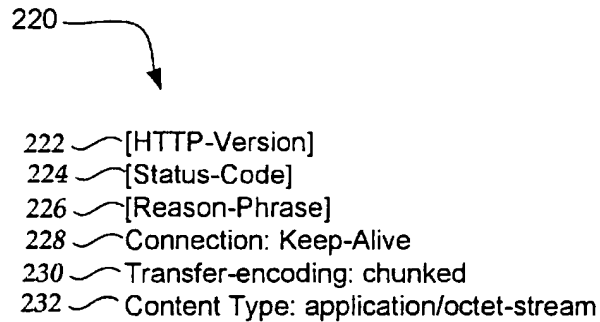


Fig.1.
PRIOR ART

*Fig.2.**Fig.5.**Fig.6.*

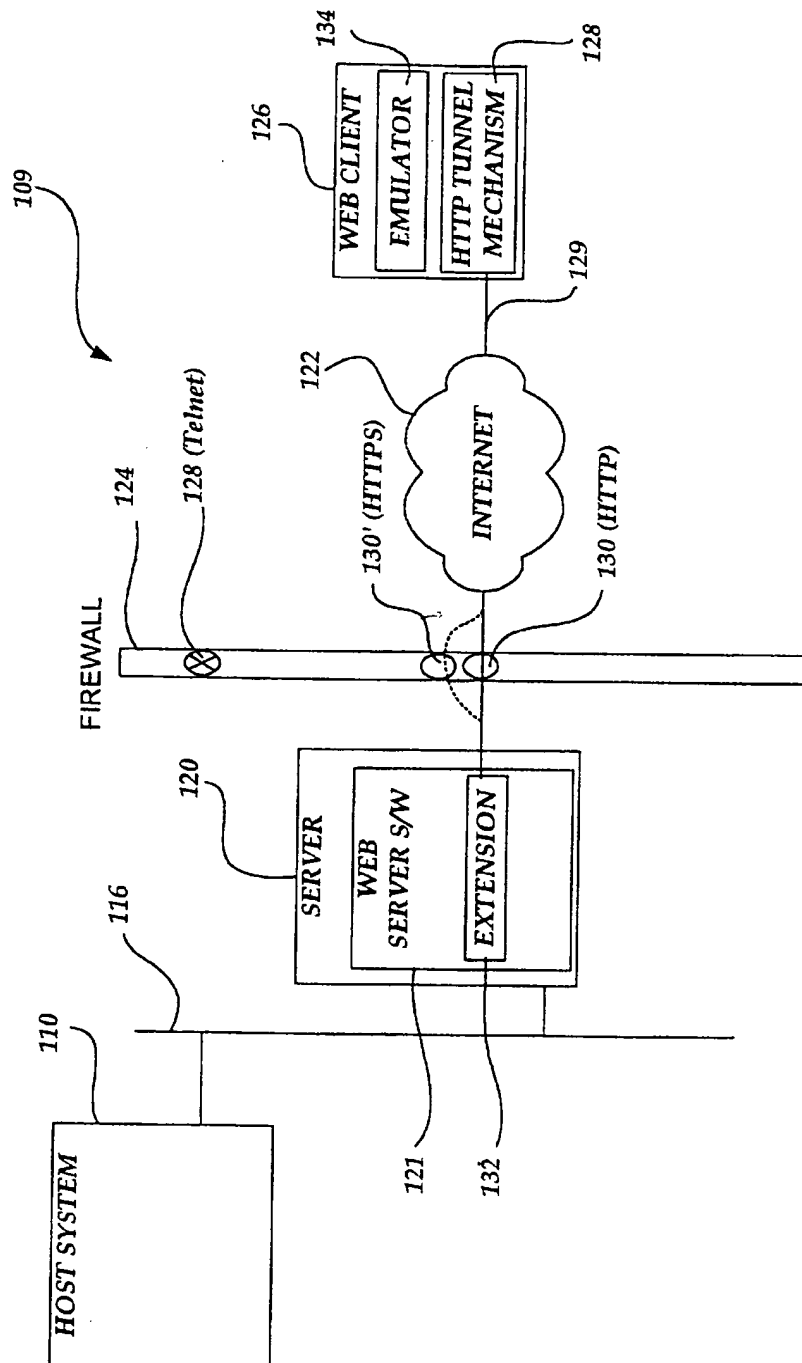


Fig.3.

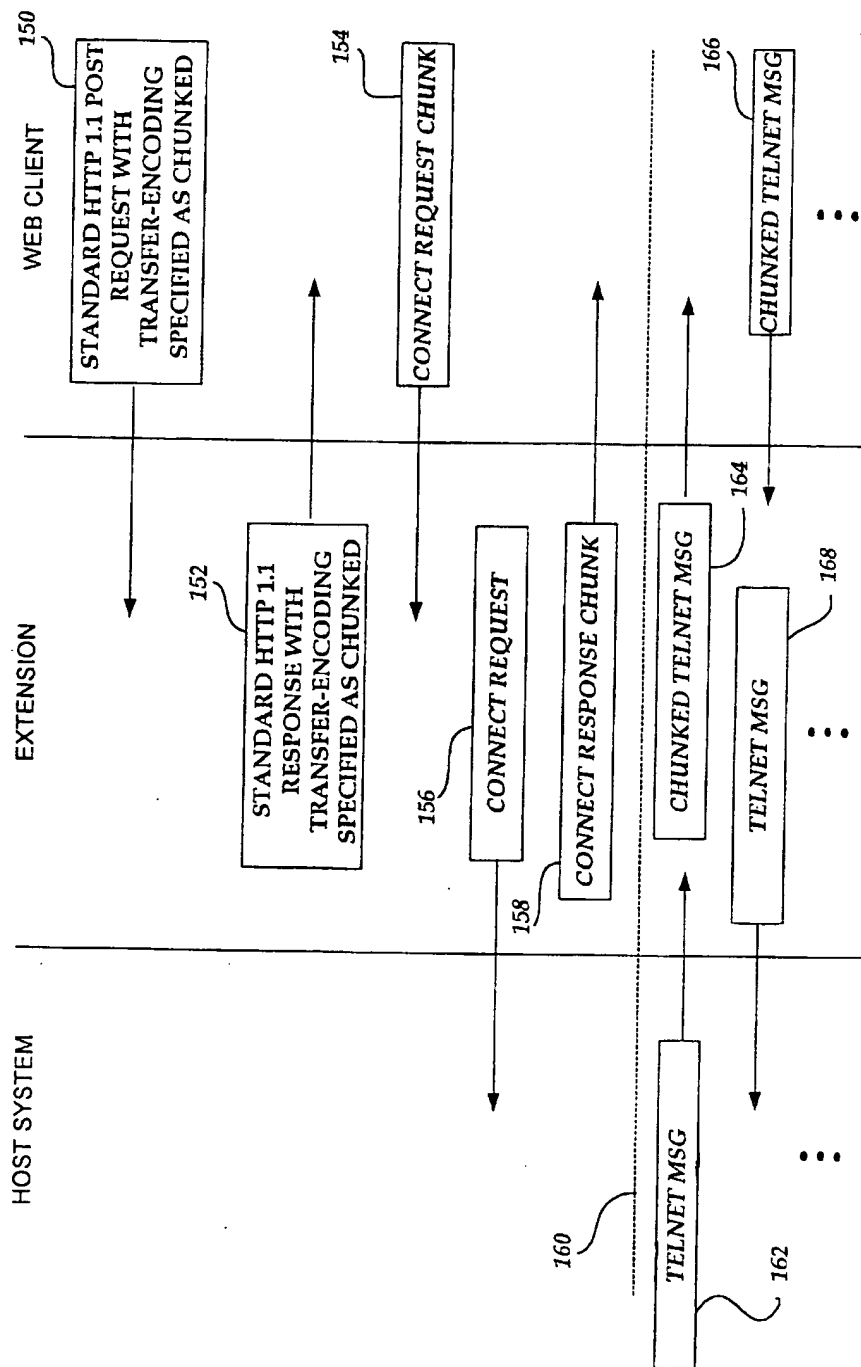


Fig. 4.

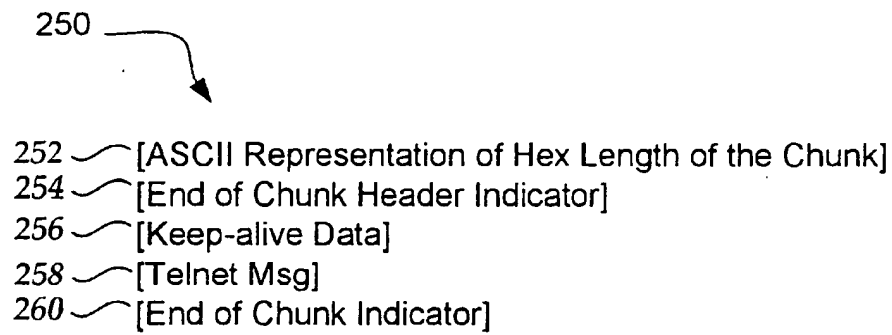


Fig.7.

PRIOR ART

1

METHOD AND SYSTEM FOR PROVIDING A PERSISTENT HTTP TUNNEL

FIELD OF THE INVENTION

The present invention relates generally to communications between two computers, and, more particularly, to a communication method and system that provides a persistent HTTP tunnel for a connection-oriented protocol between the two computers.

BACKGROUND OF THE INVENTION

Many corporations continue to maintain their corporate computer data on what are referred to as "legacy host" systems. These systems are generally older mainframe or mini-computers which cannot be easily replaced. As personal computers become more commonplace, a significant effort has been devoted to methods by which a user of a personal computer can access data stored on a legacy host. One such system is shown in FIG. 1, whereby access to a host system 10 may be through a terminal 12 directly coupled to the host system 10 that provides a local login capability. If a workstation 14 desires to connect to the host system 10 over a Local Area Network (LAN) 16, the workstation 14 runs a terminal emulator 18. A Telnet protocol is used between the host system 10 and the workstation 14 to provide the workstation 14 access to the host system 10 as if the workstation was a local terminal directly connected to the host system.

With the phenomenal growth of the Internet, a need developed to provide the ability of a user to access a host computer from anywhere in the world. A typical configuration to allow such Internet access includes a Web server 20 coupled to the host system 10 via the LAN 16. The Web server 20 is then coupled to the Internet 22 through a firewall 24. The firewall 24 enforces a security policy between a secure internal network containing the host system 10 and an untrusted network such as the Internet. The firewall may be a personal computer (PC), a router, a mainframe, a UNIX workstation, or any combination of such devices responsible for enforcing the security policy. A Web client computer 26 runs a browser program 25 to access the Web server 20 through the Internet 22.

Because the Internet 22 is an unsecured network, most firewall security policies will not allow the Web client 26 to communicate using the well-known "Telnet port 23," shown at 28. However, most firewalls allow communications through the well-known "HyperText Transfer Protocol (HTTP) port 80," shown at 30, and the secure "HTTP port 443," shown at 30'. Therefore, a current system that provides a local login experience to a Web client 26 uses one of these HTTP ports 30 30'. In this system, the Web server 20 runs a terminal emulator 32 that provides a Telnet session with the host system 10. The Web server 20 receives the Telnet data from the host system 10 and instead of displaying the data as a typical text screen will instead send the Telnet data to a translator 33. The translator 33 translates the Telnet data to HyperText Markup Language (HTML) statements that are sent to the browser program 25 running on the Web client 26. The browser 25 then translates the HTML statements and displays an HTML page on the Web client.

A problem with this type of system is that the translated HTML screen does not look sufficiently similar to a local login screen that the user would see if they were directly connected to the host system and the interaction with the HTML screen is not sufficiently similar to the interaction with the directly connected terminal. In certain situations,

2

the differences may require additional training of the users on the Web client. Another problem is that the response times between a user request on the Web client 26 and the return response from the host system is more variable in comparison with a local login response time. For example, response times may range from a second to thirty seconds using the HTML screen, in comparison with response times in the range of one second to three seconds with a directly connected terminal. The variable response times are due to the nature of the HTTP protocol.

HTTP is a request/response protocol. The Web client 26, using the browser 25, establishes a connection with the Web server 20 and sends a request to the Web server. After the Web server sends a response to the browser 25, the connection is closed. Before additional requests may be handled, a new connection must be established. Even though the newer HTTP 1.1 specification provides a keep-alive mechanism that allows one connection for multiple objects on an HTML page, the connection is closed either by the Web server or the browser after a period of inactivity. The period of inactivity may range from several seconds to a fraction of a second depending on the activity on the Web server. Many protocols, including Telnet, have insufficient transaction rates to maintain an alive connection even when the Web server is only modestly loaded. Closing the connection and establishing a new connection creates significant overhead resulting in decreased performance.

Given the shortcomings associated with the prior art method of providing access to host computer systems for Web clients, there is a need for a method that uses existing standard ports in the firewall while providing more consistent response times similar to the response times of a workstation connected through a LAN to a host system. The present invention is directed to filling this need.

SUMMARY OF THE INVENTION

In accordance with this invention, a server, a client, and a method of operation are provided for a Web client to access a host system with performance and displays comparable to the performance and displays of a workstation connected through a LAN to the host system.

In accordance with one aspect of this invention, a method of providing a persistent HTTP tunnel for a persistent virtual session is provided. A data message complying with a connection-oriented protocol is generated at an endpoint of a connection-oriented virtual session. The data message is embedded into a chunked data message complying with a chunking option of an HTTP specification. The chunked data message is transmitted between a Web client and a Web server via an HTTP connection. Upon receiving any chunked data message at the Web server, the Web server parses the chunked data message and delivers the data message to one endpoint of the connection-oriented virtual session. In the other direction, upon receiving any chunked data message at the Web client, the Web client parses the chunked data message and delivers the data message to another endpoint of the connection-oriented virtual session. The chunked data messages from the Web client are interleaved with the chunked data messages from the Web server on the persistent HTTP tunnel.

In accordance with other aspects of this invention, the connection-oriented protocol is a Telnet protocol.

In accordance with still further aspects of this invention, one endpoint of the connection-oriented virtual session is a host system.

In accordance with yet further aspects of this invention, the other endpoint of the connection-oriented session is a Web client application.

3

In accordance with still other aspects of this invention, the Web client application is a terminal emulator.

In accordance with another aspect of this invention, a method for creating a persistent tunnel between a Web client and a Web server using an HTTP protocol for providing a persistent virtual connection between a host system and the Web client is provided. A connection between the Web client and the Web server is established using a chunking option in accordance with an HTTP protocol that allows a series of messages to be sent as chunked messages. A virtual session is established between the host system and the Web client through a Web Server extension. A plurality of host messages are transmitted from the host system to the Web server extension and inserted into a chunked host message at the Web server. The Web server forwards the chunked host messages to the Web client over the connection. The Web client parses the chunked host message and delivers the host message to an application. In the other direction, a plurality of Web client messages are transmitted from the application to a tunneling mechanism on the Web client. The tunneling mechanism inserts the client message into a chunked client message and forwards the chunked client message to the Web server over the connection. The Web server forwards the chunked client message to an extension before receiving subsequent chunked client messages. The extension parses the chunked client message and delivers the client message to the host system. The chunked data messages from the Web client are interleaved with the chunked data messages from the Web server on the persistent HTTP tunnel.

In accordance with a further aspect of the present invention, a server for providing a persistent virtual session over HTTP is provided. The server includes a server software component operable to communicate via a persistent HTTP tunnel with a first endpoint of a connection-oriented session. The server also includes an extension operable to communicate with the server software component and a second endpoint of a connection-oriented session. Upon a connect request from a client, the extension establishes a connection-oriented session with the second endpoint to provide a virtual connection-oriented session between the first endpoint and the second endpoint. After the virtual connection-oriented session is established, the extension receives one or more chunked client messages from the client. The chunked messages comply with a chunking option as specified in the Hyper Text Transfer Protocol. Each chunked client message includes a chunk header and a data portion. The extension forwards the data portion to the second endpoint over the connection-oriented session. The extension also receives one or more second endpoint messages from the second endpoint and encapsulates each second endpoint message into a chunked second endpoint message. The extension then forwards the chunked second endpoint message to the client that delivers the second endpoint message of the chunked second endpoint message to the first endpoint.

In accordance with other aspects of this invention, the second endpoint is a host system.

In accordance with still further aspects of this invention, the first endpoint is a client application.

In accordance with yet still further aspects of this invention, the client application is a terminal emulator.

In accordance with a further aspect of this invention, a client having a first endpoint of a connection-oriented session having a persistent virtual session with a second endpoint over HTTP is provided. The client includes an application for sending and receiving data messages complying

4

with the connection-oriented session at the second endpoint and an HTTP tunnel mechanism. The HTTP tunnel mechanism receives the data messages generated by the application and inserts the data messages into a chunked data message complying with HTTP and transmits the chunked data messages to a Web server. The HTTP tunnel mechanism also receives chunked data messages generated by the Web server and forwards the data messages within the chunked data message to the application.

A technical advantage of the present invention is the ability to establish a connection-oriented virtual session between a host system and a Web client through the commonly available HTTP port. Both the Web server and the Web client encapsulate the connection-oriented session data, such as Telnet, into chunks that comply with the HTTP specification. The Web server transmits the HTTP response as soon as the request is received and transmits chunked messages and receives chunked messages from the Web client in an interleaving manner without completing the original request. Because the chunks are interleaved, the present invention provides a persistent bi-directional virtual connection between the host system and the Web client.

BRIEF DESCRIPTION OF THE DRAWINGS

The foregoing aspects and many of the attendant advantages of this invention will become more readily appreciated as the same become better understood by reference to the following detailed description, when taken in conjunction with the accompanying drawings, wherein:

FIG. 1 is a block diagram of a prior art method for connecting Web clients to host computer systems;

FIG. 2 is a block diagram of a protocol stack for tunneling Telnet data on HTTP;

FIG. 3 is a block diagram of a system for connecting Web clients to host systems in accordance with the present invention;

FIG. 4 is a block diagram illustrating a communication flow among a Web client, an extension residing on a Web server, and a host system in accordance with the present invention;

FIG. 5 is a format for an HTTP Post request suitable for use in the present invention;

FIG. 6 is a format for an HTTP response message to the HTTP Post request of FIG. 5 suitable for use in the present invention; and

FIG. 7 is a format of an HTTP chunk message suitable for use in the present invention.

DETAILED DESCRIPTION OF THE PREFERRED EMBODIMENT

In accordance with the present invention, access to host systems from Web clients is available in a manner that provides performance and displays comparable to the performance and displays of a workstation connected through a LAN to the host system. The present invention provides a system and method for providing a persistent HTTP tunnel for any connection-oriented protocol and keeping the HTTP tunnel connection between the Web client and the Web server persistent for the duration of the communication between the Web client and host system. By providing a persistent connection for the entire duration, the present invention achieves performance comparable to the performance of the workstation 14 connected through the LAN 16 to the host system, as shown in FIG. 1. In addition, the present invention utilizes an existing port in the firewall rather than requiring a new hole in the firewall.

5

Although the presently preferred embodiment of the invention utilizes the connection-oriented protocol Telnet, those skilled in the art will recognize that other connection-oriented protocols may be used. As mentioned earlier, the Telnet protocol is typically used between a host system and a client residing on a network to provide the client access to the host system as if the client was a local terminal directly connected to the host system.

FIG. 2 is a block diagram illustrating how Telnet data is tunneled through HTTP in accordance with the present invention. A protocol stack 100 of the present invention has a first or lower layer 102 that represents an Internet layer or a network layer that shields higher layers 104-108 from a physical network architecture. An Internet Protocol (IP) is the protocol of the first layer 102 and is a connectionless protocol that does not provide reliability, flow control or error recovery. The IP provides a routing function that ensures that messages will be correctly delivered to their destination. A second layer 104 is a transport layer that provides an end-to-end reliable data transfer. The second layer 104 allows multiple applications to be supported simultaneously. A Transmission Control Protocol (TCP) is used in the second layer 104 to provide a reliable exchange of information. A third layer 106 is an application layer. An interface between the application layer and the transport layer is defined by port numbers and sockets. In contrast to prior art methods of implementing Telnet on top of TCP, the present invention tunnels Telnet on top of the HTTP(S) protocol of the third layer 106 resulting in a fourth layer 108 of the protocol stack 100. One skilled in the art will appreciate that HTTP is considered the "official" application protocol of the World Wide Web and Telnet is considered the "official" protocol for emulating a remote terminal. The tunneling of the Telnet protocol on top of the HTTP protocol will be described in greater detail below.

FIG. 3 is a block diagram of a system for connecting Web clients to host systems in accordance with the present invention. A system 109 includes a Web server 120 coupled to a host system 110 via a LAN 116. The Web server 120 is also coupled to an Internet 122 through a firewall 124. The firewall 124 enforces a security policy between a secure internal network containing the host system 110 and an untrusted network such as the Internet. The firewall 124 may be a personal computer (PC), a router, a mainframe, a UNIX workstation, or any combination of such devices responsible for enforcing the security policy.

A Web client 126 is coupled to the Web server 120 through the Internet 122. The system 109 further comprises an extension 132 residing on the Web server 120. The extension 132 may run in the same process space as Web server software 121 or communicate with the Web server software 121 using an interprocess communication well-known in the art. The Web server software 121 passes all commands and data designating the extension 132 to the extension 132. The commands and data are received from the Web client through a HyperText Transfer Protocol (HTTP) port 80, shown at 130, or the secure HTTP port 443, shown at 130', in the firewall 124. As one skilled in the art will appreciate, the firewall 124 passes the commands and data from the Web client onto the Web server 120. Because the present invention utilizes the standard HTTP ports 130 and 130', additional holes in the firewall are not necessary. In addition, because the present invention utilizes the standard HTTP ports, the present invention minimizes the amount of development necessary to create a secure Web server environment. Rather, the present invention can utilize existing resources residing on the Web server, thereby

6

leveraging their development costs. The extension 132 is responsible for providing a virtual session between the host system 110 and the Web client 126.

The Web client 126 comprises an HTTP tunnel mechanism 128 and an emulator 134. The HTTP tunnel mechanism 128 is a software module that provides an Application Programming Interface (API) for TCP, such as Winsock created by Microsoft Corporation, of Redmond, Wash., or WDWsock created by Wall Data Incorporated, of Kirkland, Wash. The HTTP tunnel mechanism 128 receives messages created in the extension 132, parses the messages, and passes any Telnet data or commands to the emulator 134 so that the emulator 134 can emulate a host terminal.

The present invention utilizes a chunking option as specified in the HyperText Transfer Protocol specification 1.1 dated January 1997, referred to as HTTP/1.1 (RFC 2068), the specification of which is hereby incorporated by reference. The chunking option refers to a new transfer coding implementation that allows a body of a message to be transferred as a series of chunks, "chunked data," each with its own size indicator, followed by an optional footer containing entity-header fields. Prior to the chunking option, the size of the message along with the entire message must be sent at the same time. The present invention expands the use of the chunking option by sending a series of HTTP messages with embedded session-oriented data, "chunked messages," between the Web server and the Web client without sending an end chunk message. This allows one connection to remain active during the series of interleaved HTTP messages between the Web server and the Web client, thereby creating an HTTP tunnel 129 that is persistent for the duration of the communication between the host system and the Web client. The interleaved HTTP messages include messages that alternate between the Web client and the Web server as the sender for each message or messages that alternate between one or more Web client messages and then one or more Web server messages. Therefore, a persistent virtual connection between two endpoints of the connection-oriented protocol is provided. One endpoint located on the host system and the other endpoint on the Web client. In the embodiment described, a terminal emulator is the endpoint on the Web client and a Telnet process is the endpoint on the host system.

Depending upon the type of Web server 120, a setting may need to be changed to inform the Web server 120 not to process chunked data designating the extension 132. By changing the setting, the extension is allowed to handle the chunked data in a manner described in detail below. To change the setting of any Web server 120 running Microsoft Internet Information Server (IIS) 4.0 by Microsoft Corporation, the extension 132 accesses a metabase and programmatically changes an MD_UPLOAD_READAHEAD_SIZE parameter to zero (0) for the named extension 132. The MD_UPLOAD_READAHEAD_SIZE parameter is typically used to fine tune memory usage on the Web server 120. Typically, the MD_UPLOAD_READAHEAD_SIZE parameter is set to a value that is an average size of a request message received from a Web client 126. The present invention therefore modifies the MD_UPLOAD_READAHEAD_SIZE parameter for an unusual purpose of having the Web server software 121 ignore processing of chunked data that is designated to be passed to the extension 132. By disabling the processing of chunked data in the Web server software 121, the extension 132 can provide the processing for the chunked data, thereby allowing a connection-oriented protocol to be tunneled on top of HTTP and creating a bi-directional connection that

7

remains active for the duration of the communication flow among the Web client 126, the extension 132, and the host system 110.

FIG. 4 is a block diagram illustrating the communication flow among the Web client 126, the extension 132, and the host system 110. Again, Telnet is used as the connection-oriented protocol and is given by way of nonlimiting example only. At block 150, the Web client 126 generates a standard HTTP/1.1 post request with transfer-encoding specified as chunked. The creation of the post request will be described in greater detail below. At block 152, the extension receives the post request, and generates a standard HTTP/1.1 response with transfer-encoding specified as chunked which is sent back to the Web client. At block 154, the Web client generates and sends a connect request chunk to the extension. The connection request chunk includes a destination port field for specifying the port number of the host system for connection. In the above-described embodiment, the destination port field contains a value specifying the Telnet port 23. The connect request chunk also includes a destination IP address field for specifying the IP address for the host system. Because the connect request chunk occurs after the initial post response, the connect request chunk uses the format for chunked data. At block 156, the extension sends a connect request to the host system to establish a Telnet session. At block 158, the extension generates and sends a connect response chunk to the Web client. After the Web client receives the connect response chunk, the end-to-end session between the Web client and the host system via the extension is active.

The communication flow occurring after the reference line 160 represents Telnet data tunneled over HTTP. A typical exchange has the host system starting Telnet negotiation. At block 162, the host system generates a Telnet message. The Telnet message includes Telnet commands and/or Telnet data and is sent to the extension. At block 164, the extension encapsulates the Telnet message into a chunked Telnet message by embedding the Telnet message in a chunk that complies with the HTTP/1.1 format with Transfer-Encoding specified as chunked. The chunked Telnet message is sent via the HTTP tunnel 129 through the HTTP port over the Internet to the Web client.

The HTTP tunnel mechanism 128 in the Web client 126 parses the chunked Telnet message and provides the emulator 134 the original Telnet message as sent from the host system. Whenever the emulator generates a Telnet message, the HTTP tunnel mechanism 128 creates a chunked Telnet message at block 166 and forwards the chunked Telnet message through the Internet to the Web server software that in turn passes the chunked Telnet message to the extension 132. The extension unchunks the chunked Telnet message and forwards to the host system only the Telnet message as originally sent by the emulator to the HTTP tunnel mechanism, at block 168. As indicated, the sending of Telnet messages and the chunking of the Telnet message into a chunked Telnet message continues without having a new connection established between the Web client and the Web server. Because only one connection is needed during the communication flow, the present invention provides performance comparable to workstations connected through a LAN to a host system (shown in FIG. 1) and responses that are less variable than response times of prior art methods using translated HTML statements.

One aspect of the present invention is enabling the Web server software 121 to pass the chunked messages through to the extension without waiting for additional chunked messages prior to delivery to the extension 132. This allows the present invention to tunnel Telnet messages on top of the

8

HTTP protocol using the standard HTTP ports 130, 130' and to use the HTTP ports 130, 130' as a bi-directional connection for a virtual session between the host system and the Web client.

FIGS. 5-7 illustrate formats for the communication flows of FIG. 4 in more detail. First, FIG. 5 illustrates a standard HTTP/1.1 post request format suitable for use in the present invention. The format of the post request is as specified in the HyperText Transfer Protocol HTTP/1.1 specification. The present invention is concerned with the contents of the post request rather than the format of the post request. At line 201, a request-URI field contains a name for the extension 132 in the Web server. The name in the request-URI field may include a dynamic link library (DLL) or other name of a component depending on the operating system platform. At line 202, an HTTP-version field must specify version 1.1 or greater. Only version 1.1 or greater allows a transfer-encoding to be chunked as is necessary for the present embodiment of the invention. On line 204, the transfer-encoding is designated as chunked.

As illustrated in FIG. 4, at blocks 154, 158, 164, chunking allows the Web client and the host system to exchange a series of messages without having to open a new connection. At line 206 of FIG. 5, in the present embodiment, the Web server is requested to accept application/octet-stream data. This is not necessary for the purposes of the present invention but is typical for Web servers to expect binary data. At line 208, in the User Agent field, an identifier for the HTTP tunnel mechanism is provided. The identifier is shown as a non-limiting example and is not necessary to practice the present invention. At line 210, in the Host field, the name of the Web server is specified. At line 212, in the Connection field, Keep-Alive is specified so that the connection remains active. However, even with Keep-Alive specified, the connection may be terminated by either the Web client or the Web server after even very brief periods of idleness. In order to achieve a true Keep-Alive connection, the present invention incorporates additional Keep-Alive data in the tunneled data as will be described in greater detail with reference to FIG. 7.

FIG. 6 illustrates a format for a standard HTTP/1.1 response. Again, the format of the data is as specified in the HTTP/1.1 specification and the present invention is only concerned with the content of the fields. Lines 222, 224, and 226 are generated by the Web server in response to the post request (FIG. 5) that was received and are explained in the HTTP/1.1 specification. Lines 228, 230, and 232 are generated by the extension before the response is sent to the Web client. At line 228, the connection is specified as Keep-Alive as described earlier. Transfer encoding is specified as chunked and the Web server indicates the format of response data as application/octet-stream (i.e., binary data rather than text or other special format data).

FIG. 7 illustrates a format for a chunk suitable for use in the present invention. At line 252, the first field specifies an ASCII representation of the hex length of the chunk. According to the HTTP 1.1 specification, by allowing each chunk to specify the hex length of the chunk, multiple chunks (messages) may be sent for one HTTP request. However, the last message must provide a unique identifier to indicate when the HTTP request is complete. Typically, the Web server software waits until all the chunks for one message body are received before generating a response. However, because the present invention uses the chunking option in a new manner and does not send a last message, the present invention prevents the Web server software from buffering the chunked messages and instead forces the Web

9

server software to send the chunked messages to the extension as the chunked messages are received. This allows the present invention to provide bi-directional communication without waiting for an entire message and without establishing more than one connection. At line 254, an end of chunk header indicator is chosen according to the HTTP/1.1 specification. Lines 256 and 258 provide the tunneling aspect of the present invention. First, line 256 provides a data field that may be sent in all chunked packets to keep the connection alive. This Keep-Alive field is necessary because some Web servers will close a connection if additional chunks are not received within a certain time frame. Therefore, in the present invention, the HTTP tunnel mechanism that is running on the Web client generates a chunk with only the Keep-Alive data field and without any additional Telnet data after a predetermined period of inactivity. The sending of only the Keep-Alive data in a chunk allows the connection to remain alive even during periods of inactivity. Line 258 represents the tunneling of the Telnet messages generated by either the host system or the Web client emulator. The Telnet messages generated by either the host system or the Web client emulator are written into the Telnet message field. Line 260 specifies the end of chunk indicator as specified in the HTTP/1.1 specification.

The connection between the Web client and the host system is terminated when the Web client closes the HTTP tunnel mechanism 128. The extension will also recognize when the host system has prematurely ended the session and will allow the Web server to close the HTTP tunnel.

Although the present embodiment utilizes an emulator that runs as a stand-alone program on the Web client, it will be appreciated that the functionality of the emulator and/or the HTTP tunnel mechanism may be incorporated into a browser. In another embodiment, the Web client may initiate a HEAD response/request before establishing a connection. The initiation of a HEAD response/request is necessary when a Web server needs authentication of the Web client access. The HEAD response provides the Web client with information about authentication that the Web client may incorporate into the post request and the subsequent chunked data.

As one skilled in the art will appreciate, Telnet has many variations such as TN5250, NVT, VT220, and TN3270. These and other proprietary protocols may be used without departing from the scope of this invention. Likewise, the host session may provide a terminal session such as a 5250 type terminal session, a NVT type terminal session, a VT220 type terminal session, and a 3270 type terminal session without departing from the scope of this invention. In addition, although the present embodiment included an IBM host system, it will be appreciated that other host systems, such as Hewlett-Packard and UNIX host systems may be used. Further, even though the embodiment shown provides a virtual session through the Internet, the Internet may be replaced with an intranet, WAN, or other network using HTTP.

While the preferred embodiment of the invention has been illustrated and described, it will be appreciated that various changes can be made therein without departing from the spirit and scope of the invention.

The embodiments of the invention in which an exclusive property or privilege is claimed are defined as follows:

1. A method of providing a persistent HTTP tunnel for a persistent virtual session, the method comprising:

- (a) generating a data message complying with a connection-oriented protocol;

10

- (b) embedding the data message within a chunked data message complying with a chunking option of a HTTP specification;

- (c) transmitting the chunked data message between a Web client and a Web server via an HTTP connection;

- (d) upon receiving any chunked data message at the Web server, parsing the chunked data message and delivering the data message to one endpoint of a connection-oriented session, and upon receiving any chunked data message at the Web client, parsing the chunked data message and delivering the data message to another endpoint of the connection-oriented session; and

- (e) interleaving the chunked data messages from the Web client with the chunked data messages from the Web server on the persistent HTTP tunnel.

2. The method of claim 1, wherein the connection-oriented protocol is a Telnet protocol.

3. The method of claim 1, wherein the one endpoint of the connection-oriented session is a host system.

4. The method of claim 3, wherein the other endpoint of the connection-oriented session is a Web client application.

5. The method of claim 4, wherein the Web client application is a terminal emulator.

6. The method of claim 1, further comprising (f) transmitting a keep alive chunk message by the Web client to the Web server after a pre-determined time period in which chunked data messages were not transmitted.

7. A method for creating a persistent tunnel between a Web client and a Web server using an HTTP protocol for providing a persistent virtual connection between a host system and the Web client, the method comprising:

- establishing a connection between the Web client and the Web server using a chunking option in accordance with an HTTP protocol that allows a series of messages to be sent as chunked messages;

- establishing a virtual session between a host system and the Web client through an extension;

- transmitting a plurality of host messages from the host system to the Web server, inserting the host message within a chunked host message at the Web server, forwarding the chunked host message to the Web client over the connection, parsing the chunked host message at the Web client and delivering the host message to an application;

- transmitting a plurality of Web client messages from the application to a tunneling mechanism on the Web client, inserting the client message within a chunked client message at the Web client, forwarding the chunked client message to the Web server over the connection, parsing the chunked client message at the Web server and delivering the client message to the host system; and

- interleaving the chunked data messages from the Web client with the chunked data messages from the Web server on the persistent HTTP tunnel;

- wherein the transmitted chunked client message is forwarded to the extension before the Web server is forwarded the entire plurality of chunked client messages.

8. The method of claim 7, wherein the application is a terminal emulator.

* * * * *



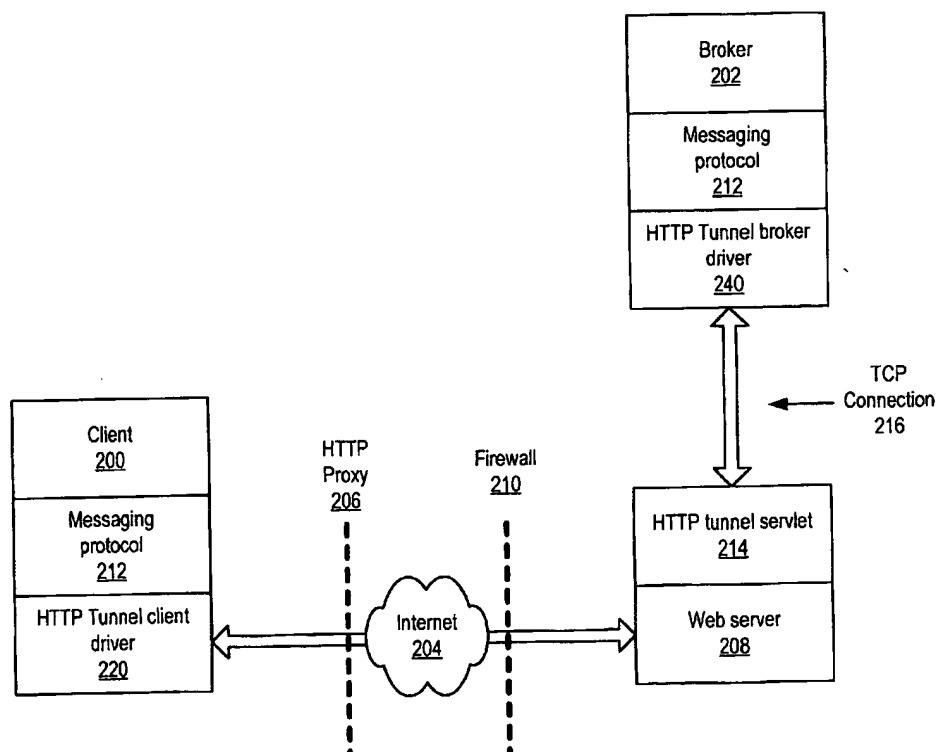
US 20030009571A1

(19) **United States**(12) **Patent Application Publication** (10) **Pub. No.: US 2003/0009571 A1****Bavadekar**(43) **Pub. Date:****Jan. 9, 2003**(54) **SYSTEM AND METHOD FOR PROVIDING
TUNNEL CONNECTIONS BETWEEN
ENTITIES IN A MESSAGING SYSTEM**(76) **Inventor: Shailesh S. Bavadekar, Sunnyvale, CA
(US)**

Correspondence Address:
Robert C. Kowert
Conley, Rose, & Tayon, P.C.
P.O. Box 398
Austin, TX 78767-1400 (US)

(21) **Appl. No.: 09/894,318**(22) **Filed: Jun. 28, 2001****Publication Classification**(51) **Int. Cl.⁷ G06F 15/16**(52) **U.S. Cl. 709/230; 709/203**(57) **ABSTRACT**

A system and method for providing HTTP tunnel connections between entities such as clients and servers in a messaging system is described. An HTTP tunnel connection layer is described that may be used to provide reliable, full duplex virtual connections between entities (e.g. clients and brokers) in a distributed application environment using a messaging system. Also described is a novel HTTP tunneling protocol that may be used by the HTTP tunnel connection layer. The HTTP tunnel connection layer may be used by clients to access messaging servers through proxy servers and firewalls, thus expanding the scope of from where clients can access brokers. Using this layer, brokers as well as clients may initiate messaging system messages. This layer may also provide guaranteed data delivery with correct sequencing even in case of a failure on the network. This layer may also provide end-to-end flow control.



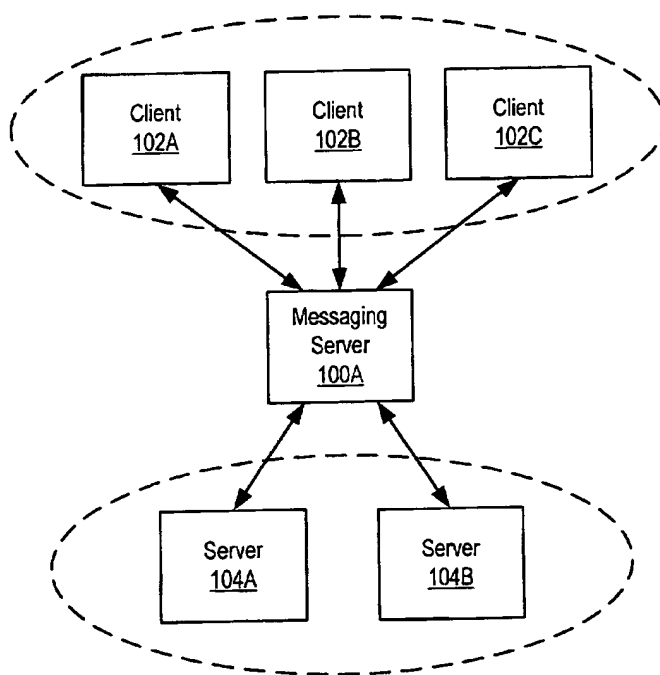


Figure 1 - Prior Art

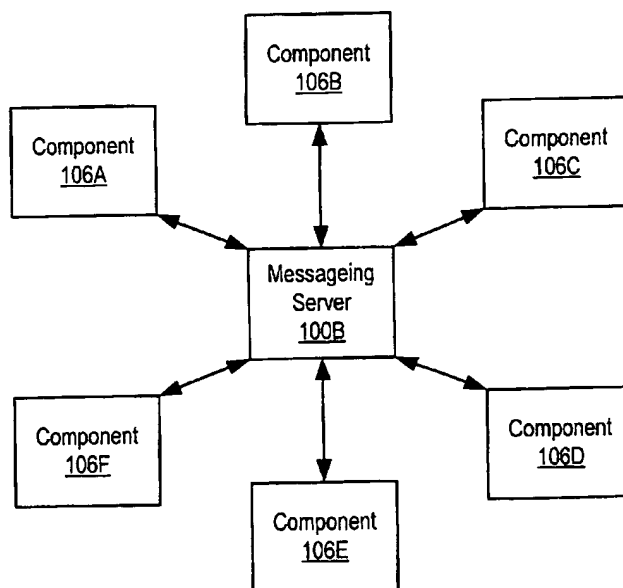


Figure 2 - Prior Art

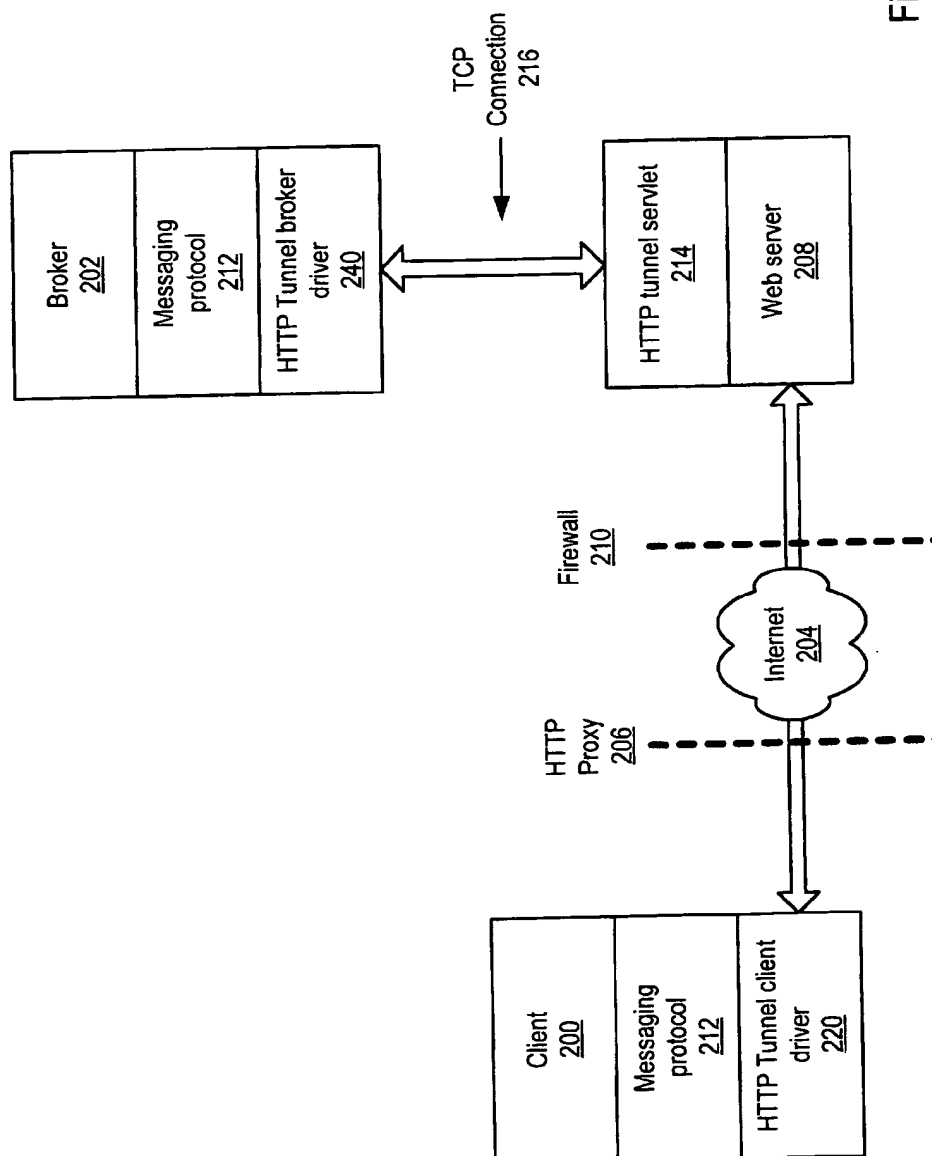


Figure 3A

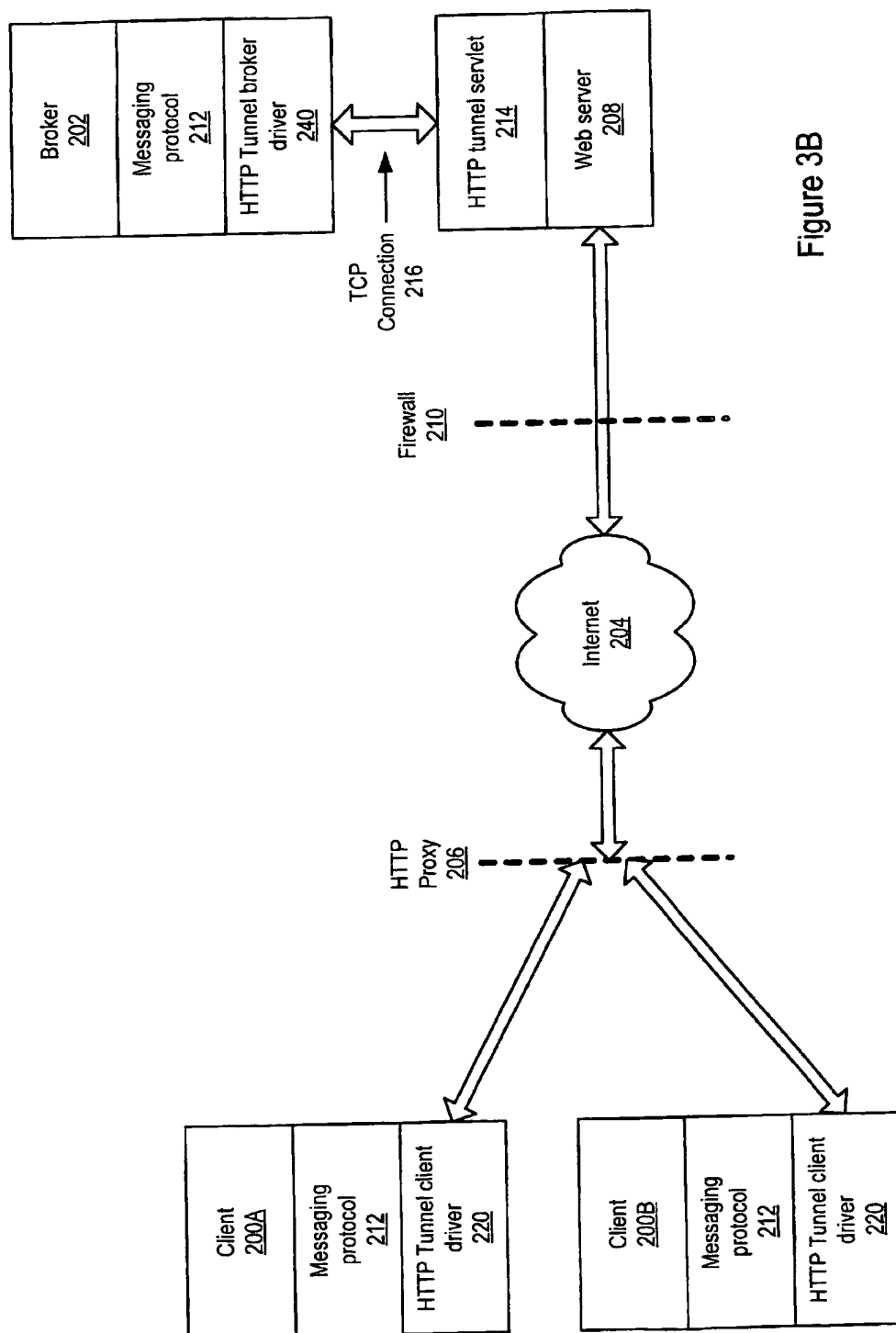


Figure 3B

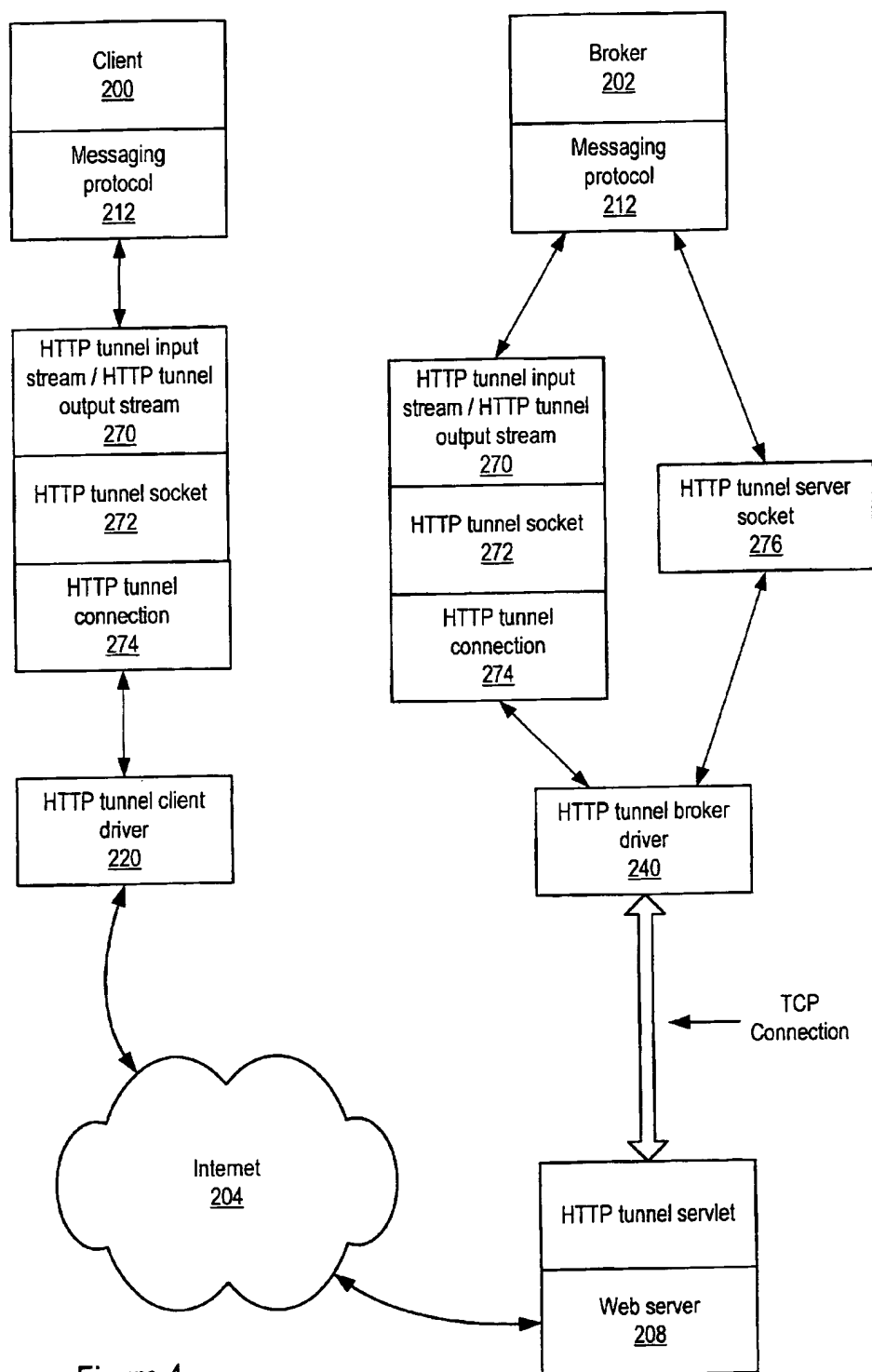


Figure 4

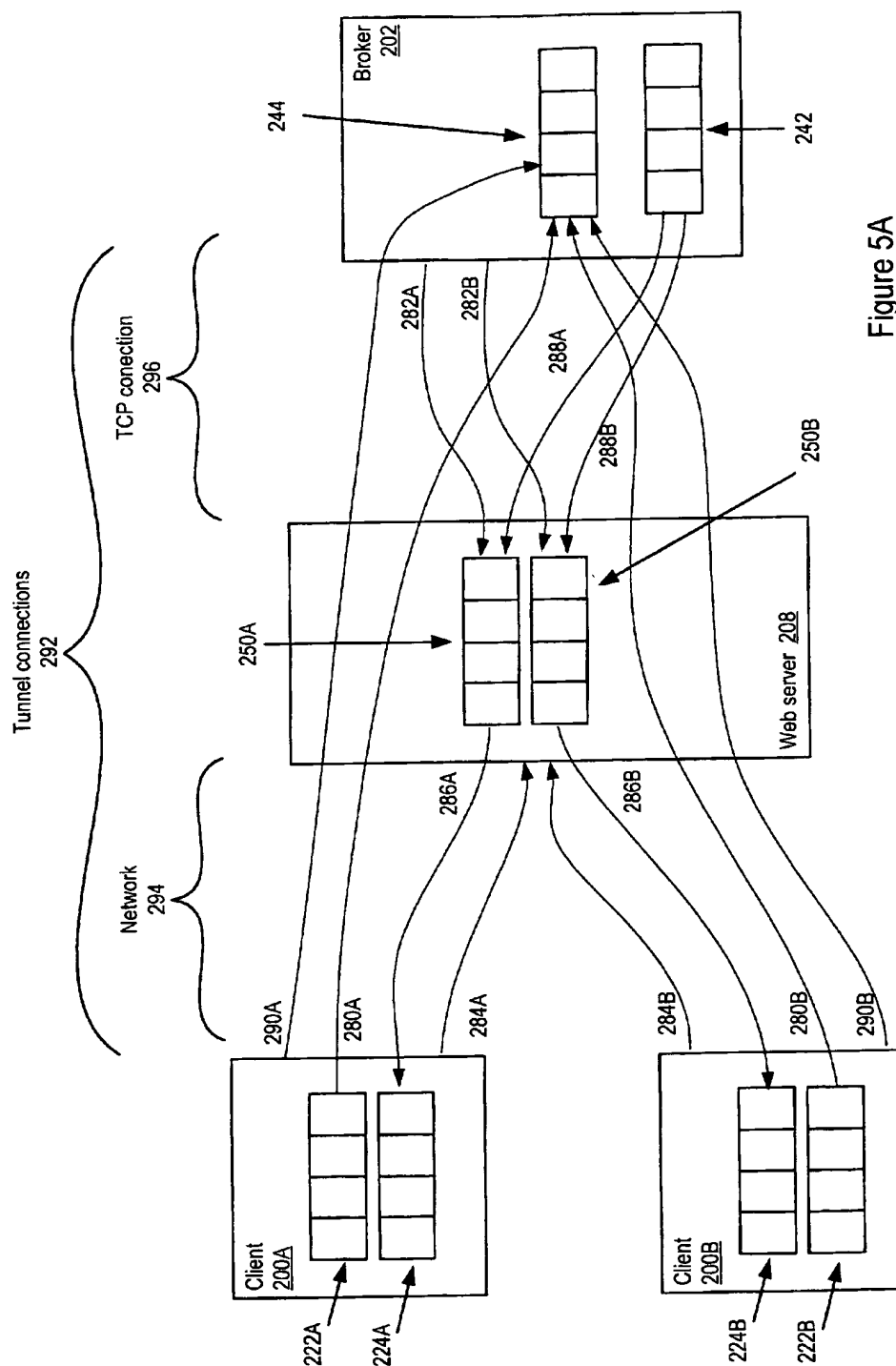


Figure 5A

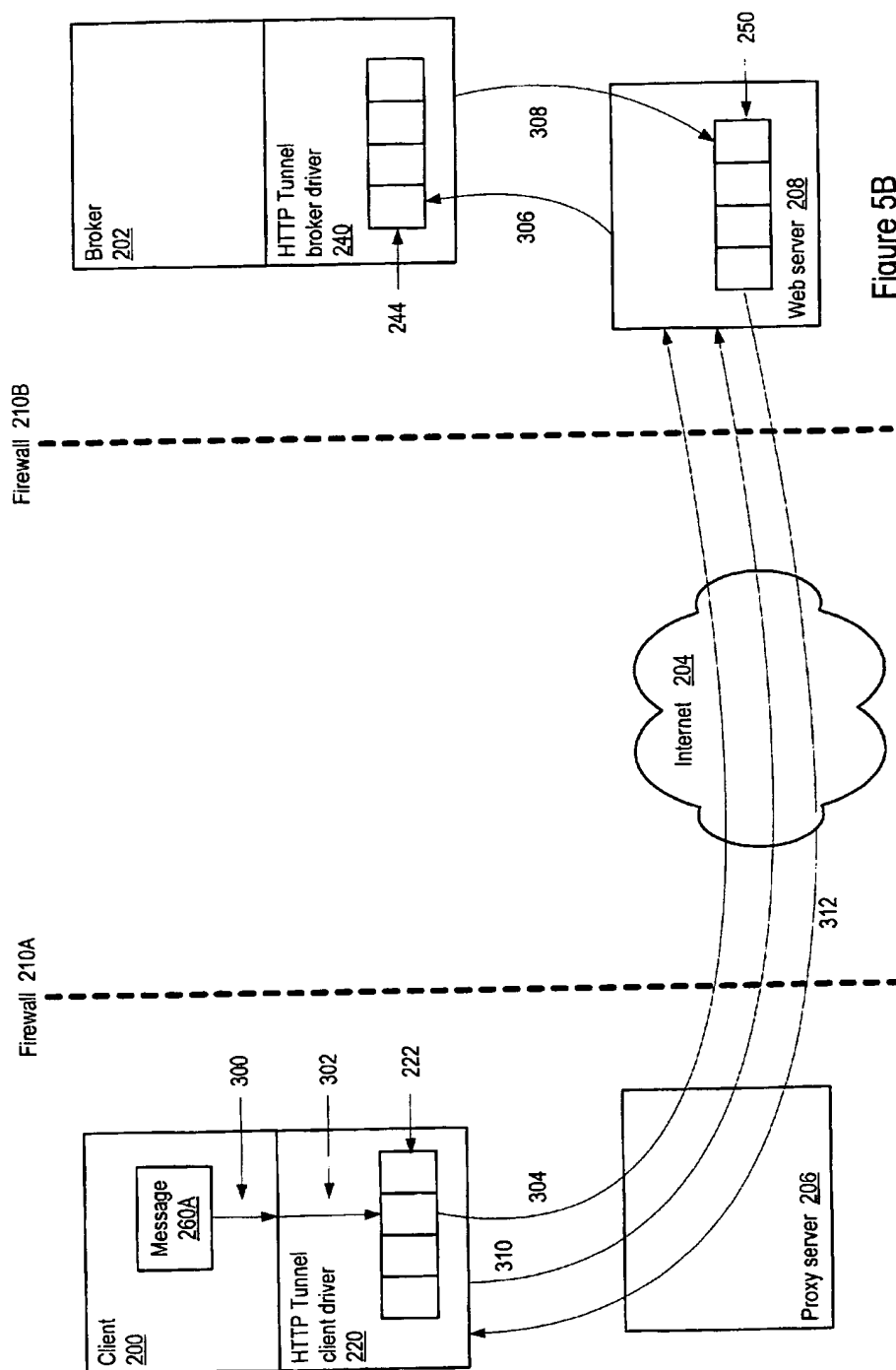


Figure 5B

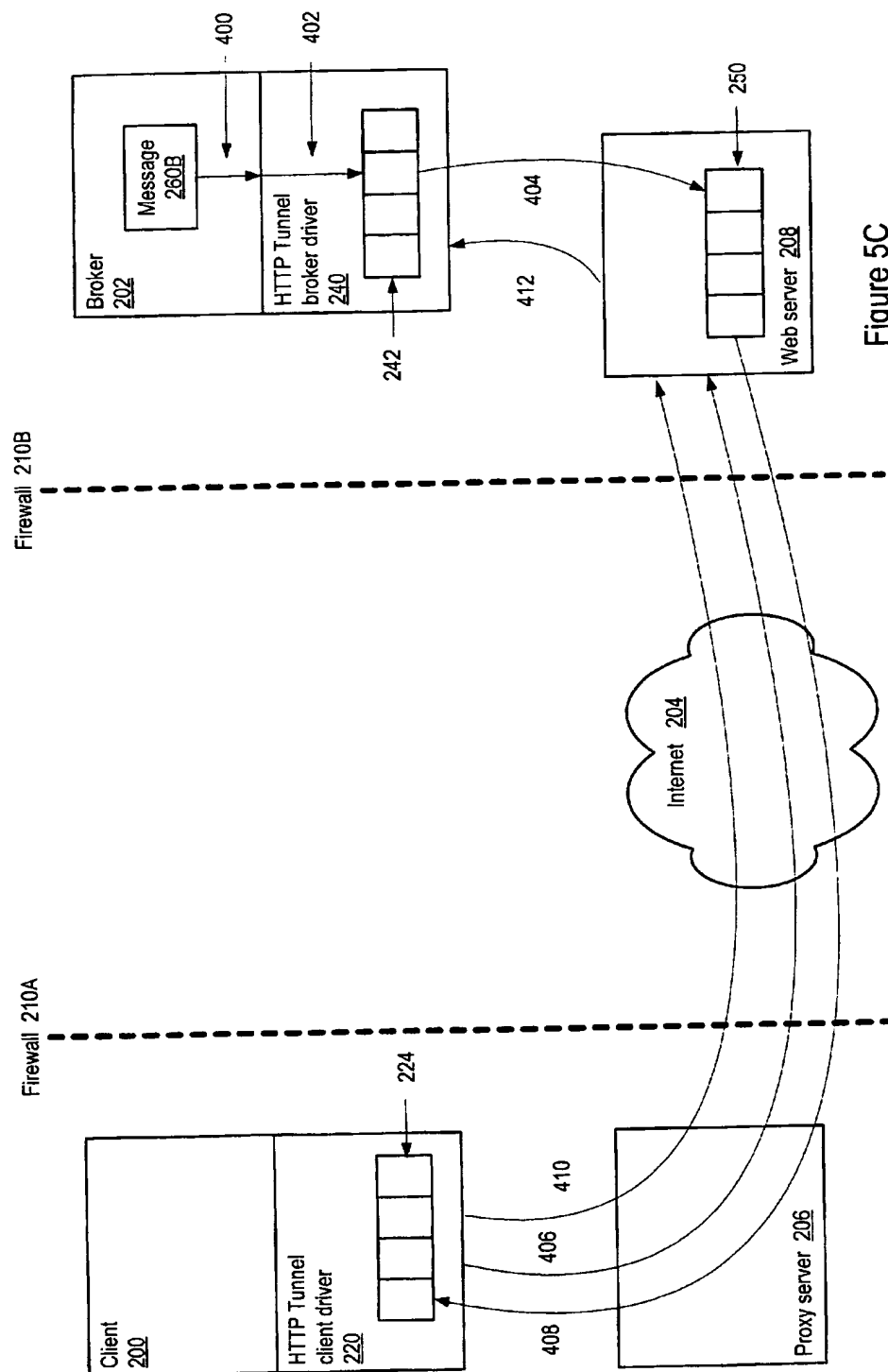


Figure 5C

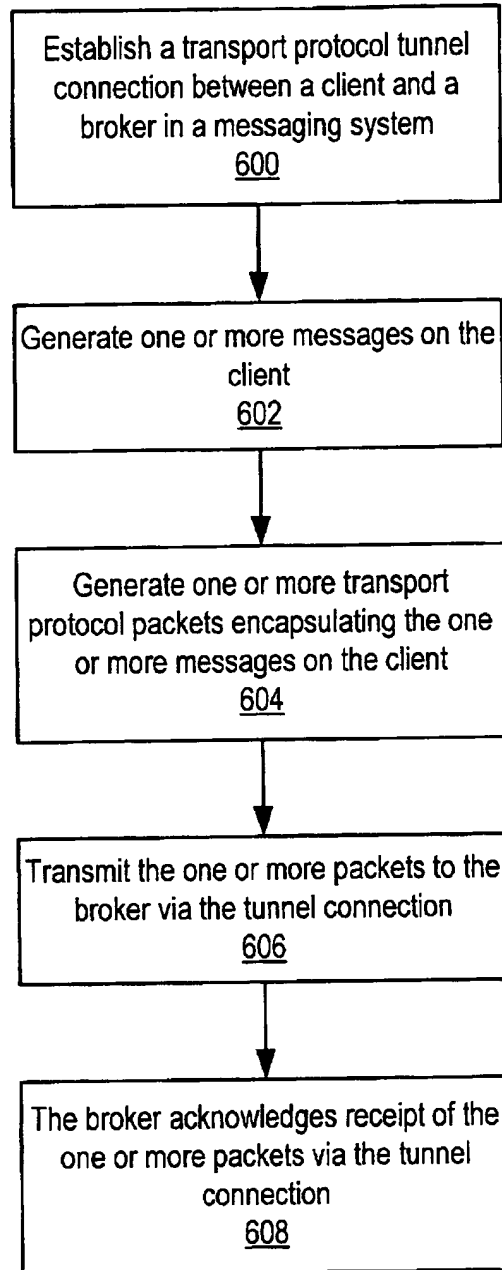


Figure 6A

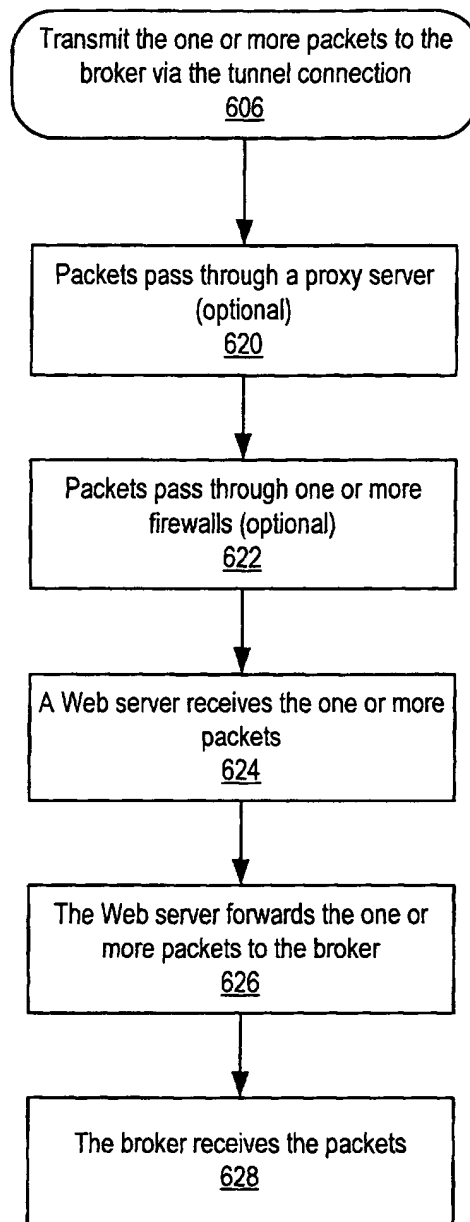


Figure 6B

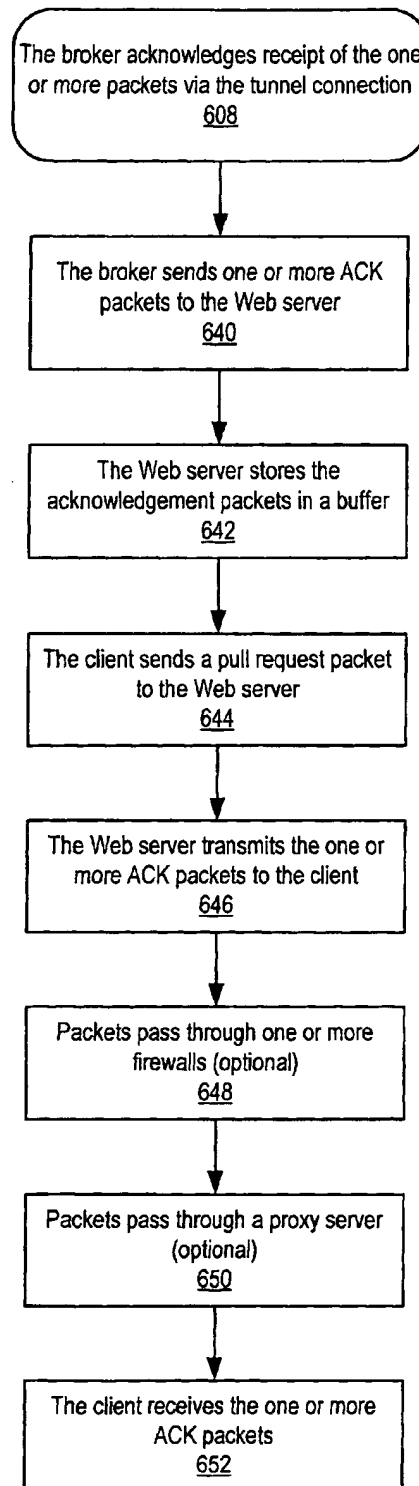


Figure 6C

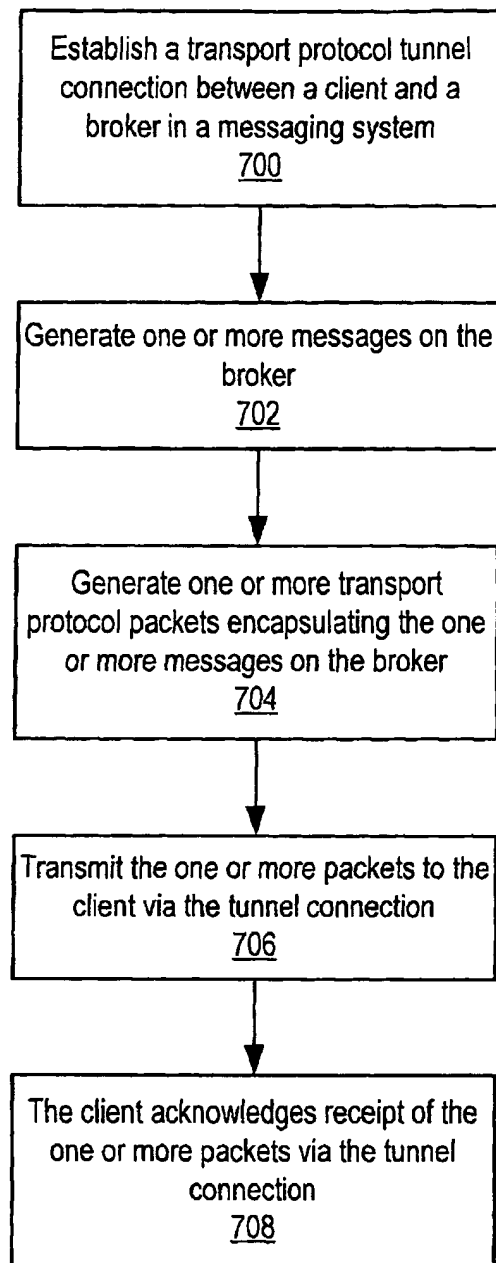


Figure 7A

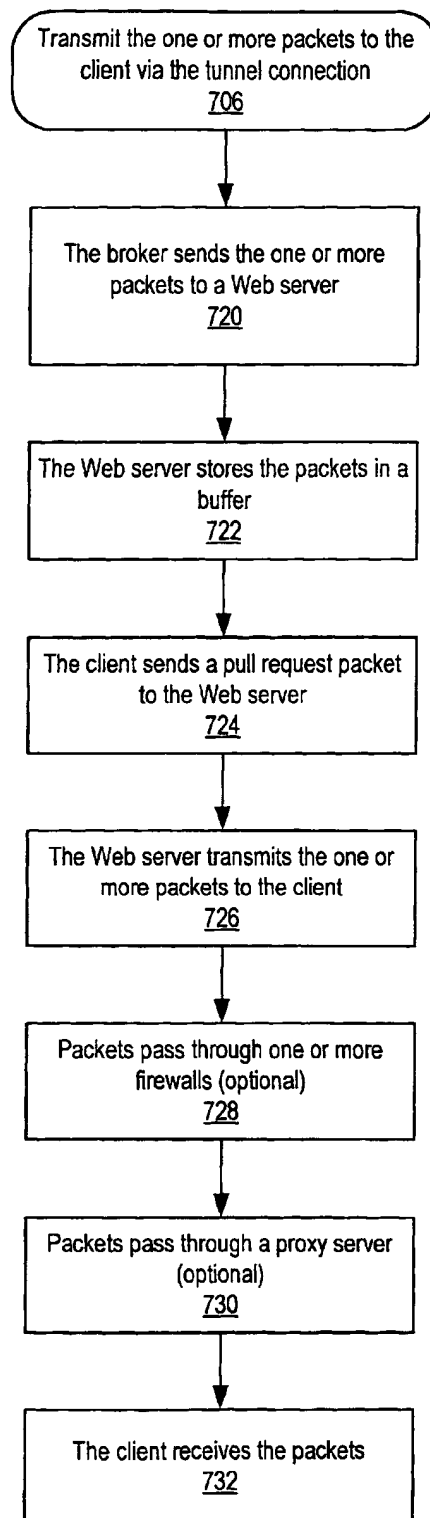


Figure 7B

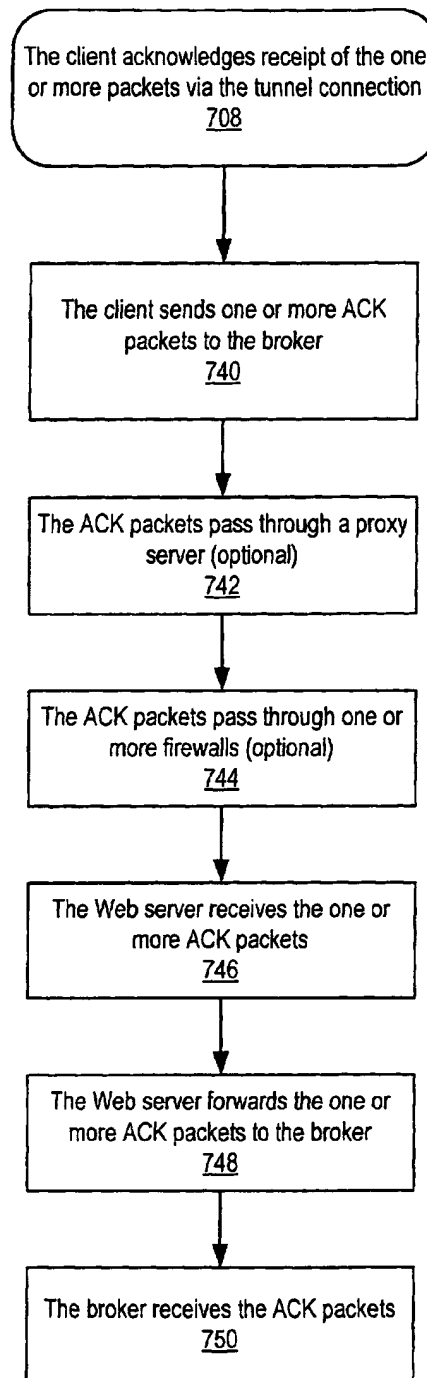


Figure 7C

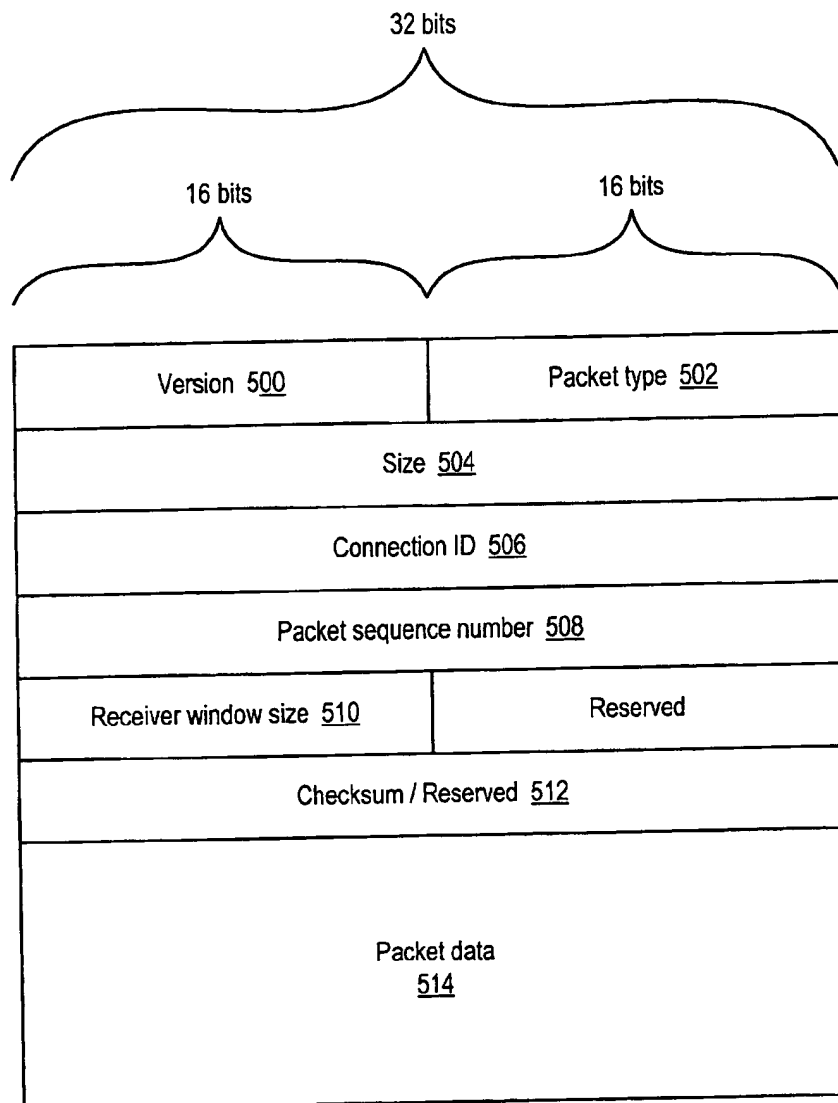


Figure 8

Exemplary tunneling packet format

SYSTEM AND METHOD FOR PROVIDING TUNNEL CONNECTIONS BETWEEN ENTITIES IN A MESSAGING SYSTEM

BACKGROUND OF THE INVENTION

[0001] 1. Field of the Invention

[0002] This invention relates to computers and networks of computers, and more particularly to a system and method for providing transport protocol tunnel connections between entities or nodes such as clients and servers in a messaging system.

[0003] 2. Description of the Related Art

[0004] Messaging is playing an increasingly important role in computing. Its advantages are a natural result of several factors: the trend toward peer-to-peer computing, greater platform heterogeneity, and greater modularity, coupled with the trend away from synchronous communication between processes. The common building block of a messaging service is the message. Messages are specially formatted data describing events, requests, and replies that are created by and delivered to computer programs. Messages contain formatted data with specific meanings. Messaging is the exchange of messages to a messaging server, which acts as a message exchange program for client programs. A messaging server is a middleware program that handles messages that are sent by client programs for use by other programs. Typically, client programs access the functionality of the messaging system using a messaging application program interface (application program interface). A messaging server can usually queue and prioritize messages as needed, and thus saves each of the client programs from having to perform these services. Rather than communicate directly with each other, the components in an application based around a message service send messages to a message server. The message server, in turn, delivers the messages to the specified recipients

[0005] There are two major messaging system models: the point-to-point model and the publish and subscribe model. Messaging allows programs to share common message-handling code, to isolate resources and interdependencies, and to easily handle an increase in message volume. Messaging also makes it easier for programs to communicate across different programming environments (languages, compilers, and operating systems) since the only thing that each environment needs to understand is the common messaging format and protocol. The messages involved exchange crucial data between computers—rather than between users—and contain information such as event notification and service requests. IBM's MQSeries and iPlanet Message Queue are examples of products that provide messaging interfaces and services.

[0006] FIG. 1 illustrates a typical messaging-based application. This application is a modification of the traditional client/server architecture. The major difference is the presence of a messaging server 100A between client 102 and server 104 layers. Thus, rather than communicating directly, clients 102 and servers 104 communicate via the messaging server 100A. The addition of the messaging server 100A adds another layer to the application, but it greatly simplifies the design of both the clients 102 and the servers 104 (they are no longer responsible for handling communications

issues), and it also enhances scalability. Note that servers in a messaging system may also be referred to as "brokers".

[0007] FIG. 2 illustrates another messaging-based application based on point-to-point architecture. This type of application almost demands a centralized messaging server 100B. Without one, each component 106 would be responsible for creating and maintaining connections with the other components 106. A possible alternative approach would be to architect the system around a communication bus, but this would still leave each component 106 in charge of message delivery issues.

[0008] Java Message Service (JMS)

[0009] Java Message Service (JMS) is an application program interface (API) from Sun Microsystems that supports messaging between computers in a network. JMS provides a common interface to standard messaging protocols and also to special messaging services in support of Java programs. Sun advocates the use of the JMS for anyone developing Java applications, which can be run from any major operating system platform. Using the JMS interface, a programmer can invoke the messaging services of IBM's MQSeries, Progress Software's SonicMQ, and other messaging product vendors.

[0010] The JMS API may:

- [0011] Provide a single, unified message API
- [0012] Provide an API suitable for the creation of messages that match the format used by existing, non-JMS applications
- [0013] Support the development of heterogeneous applications that span operating systems, platforms, architectures, and computer languages
- [0014] Support messages that contain serialized Java objects
- [0015] Support messages that contain eXtensible Markup Language (XML) pages
- [0016] Allow messages to be prioritized
- [0017] Deliver messages either synchronously or asynchronously
- [0018] Guarantee messages are delivered once and only once
- [0019] Support message delivery notification
- [0020] Support message time-to-live
- [0021] Support transactions

[0022] The JMS API is divided into two nearly identical pieces. One implements a point-to-point model of messaging, and the other implements a publish and subscribe model of messaging. Each of these models is called a domain. The APIs are almost identical between the domains. The separation of the API into two domains relieves vendors that support only one messaging model from providing facilities their product doesn't natively support.

[0023] Enterprise Messaging Systems

[0024] Enterprise messaging systems may be developed using a messaging service such as JMS. An enterprise messaging system may be used to integrate distributed,

loosely coupled applications/systems in a way that provides for dynamic topologies of cooperating systems/services. Enterprise messaging systems typically need to address common messaging related problems such as:

[0025] Guaranteed message delivery (e.g. persistence, durable interests, "at least once" and "once and only once" message delivery guarantees, transactions etc). Messages from one component to another must not be lost due to network or system failure. This means the system must be able to guarantee that a message is successfully delivered.

[0026] Asynchronous delivery. For large numbers of components to be able to exchange messages simultaneously, and support high density throughputs, the sending of a message cannot depend upon the readiness of the consumer to immediately receive it. If a consumer is busy or offline, the system must allow for a message to be sent and subsequently received when the consumer is ready. This is known as asynchronous message delivery, popularly known as store-and-forward messaging.

[0027] Various message delivery models (e.g. publish and subscribe or point-to-point).

[0028] Transport independence.

[0029] Leveraging an enterprise messaging system in developing business solutions allows developers to focus on their application/business logic rather than on implementing the underlying messaging layer.

[0030] iPlanet E-Commerce Solutions' iMQ (iplanet Message Queue), formerly offered by Sun Microsystems as JMQ (Java Message Queue) is an example of an enterprise messaging system, and was developed to be JMS-compliant. iMQ may use a "hub and spoke" architecture. Clients use an iMQ client library to exchange messages with an iMQ message server (also referred to as a "broker").

[0031] In an enterprise messaging system, clients exchange messages with a messaging server using a message exchange protocol. The messaging server then may route the messages based upon properties of the messages. Typically, the message exchange protocol requires a direct, fully bi-directional reliable transport connection between the client and the messaging server, such as a TCP (Transport Control Protocol) or SSL (Secure Sockets Layer) connection, which can be used only if the client and the messaging server both reside on the "intranet" (i.e. on the same side of a firewall).

[0032] Hypertext Transfer Protocol

[0033] The Hypertext Transfer Protocol (HTTP) is a set of rules for exchanging files (text, graphic images, sound, video, and other multimedia files) on the World Wide Web. HTTP may also be used on an intranet. Relative to the TCP/IP suite of protocols (which are the basis for information exchange on the Internet), HTTP is an application protocol that is implemented over TCP/IP. HTTP was designed as a stateless request-response mechanism.

[0034] A Web server is a program that, using the client/server model and HTTP, serves the files that form Web pages to Web users (whose computers contain HTTP clients that forward their requests). Every computer on the Internet that

contains a Web site must have a Web server. A Web server machine may include, in addition to the Hypertext Markup Language (HTML) and other files it can serve, an HTTP daemon, a program that is designed to wait for HTTP requests and handle them when they arrive. A Web browser is an example of an HTTP client that sends requests to server machines. When a browser user enters file requests by either "opening" a Web file by typing in a URL (Uniform Resource Locator) or clicking on a hypertext link, the browser builds an HTTP request and sends it to the Internet Protocol (IP) address indicated by the URL. The HTTP daemon in the destination server machine receives the request and, after any necessary processing, the requested file is returned.

[0035] Tunneling

[0036] Tunneling may be defined as the encapsulation of a protocol A within protocol B, such that A treats B as though it were a data link layer. Tunneling may be used to get data between administrative domains that use a protocol that is not supported by the Internet connecting those domains. A "tunnel" is a particular path that a given message or file might travel through the Internet.

[0037] Proxy Servers and Firewalls.

[0038] In an enterprise that uses the Internet, a proxy server is a server that acts as an intermediary between a workstation user and the Internet so that the enterprise can ensure security, administrative control, and caching service. A proxy server may be associated with or part of a gateway server that separates the enterprise network from the outside network and a firewall server that protects the enterprise network from outside intrusion.

[0039] A firewall is a set of related programs, usually located at a network gateway server, that protects the resources of a private network from users from other networks. An enterprise with an intranet that allows its workers access to the wider Internet installs a firewall to prevent outsiders from accessing its own private data resources and for controlling what outside resources its own users have access to.

SUMMARY OF THE INVENTION

[0040] A system and method for providing transport protocol tunnel connections between entities such as clients and servers in a messaging system is described. A transport protocol tunnel connection layer is described that may be used to provide reliable, full duplex virtual connections between entities (e.g. clients and servers) in a distributed application environment using a messaging system. This layer may be used by clients to access messaging servers (referred to as brokers) through proxy servers and firewalls, thus expanding the scope of from where clients can access brokers. This layer allows for information flow in both directions between the client and the server. Using this layer, brokers as well as clients may initiate messages in the distributed application environment. This layer may also provide guaranteed data delivery with correct sequencing even in case of a failure on the network. This layer may also provide end-to-end flow control.

[0041] The transport protocol tunnel connection layer may provide a connection-oriented, bi-directional byte stream service between nodes in a messaging system. Application data, including messages, may be carried as transport pro-

protocol packet payloads. The transport protocol tunnel connection layer allows messaging system clients to access messaging system brokers using the transport protocol instead of using direct TCP connections. This enables the clients to access the brokers through firewalls. Also, with the help of a transport protocol proxy, the clients may access the messaging service even when there is no direct IP connectivity with the broker.

[0042] Using embodiments of the transport protocol tunnel connection layer, a transport protocol tunnel connection between the client and the broker in the messaging system may be established. The client may then generate one or more messaging system messages. In one embodiment, the generated messages may then be stored in a client-side transmit buffer. One or more transport protocol packets encapsulating the one or more messages may then be generated on the client. In one embodiment, a client-side tunnel connection driver may generate the packets and include the messages as payloads of the packets. The one or more transport protocol packets may then be transmitted to the broker via the tunnel connection. In one embodiment, the client-side tunnel connection driver may handle the transmission of the packets.

[0043] A Web server may then receive the one or more transport protocol packets. The Web server may then forward the received transport protocol packets to the broker. In one embodiment, the packets may be forwarded to the broker via a TCP connection serving as one segment of the transport protocol tunnel connection between the Web server and the broker. In one embodiment, a transport protocol tunnel servlet on the Web server node may serve as an interface between the Web server and the TCP connection.

[0044] The broker may receive the transport protocol packets from the Web server. In one embodiment, a broker-side transport protocol tunnel driver may receive the packets. In one embodiment, the broker may extract the messaging system messages from the transport protocol packets and store the messages in a broker-side receive buffer. In another embodiment, the entire transport protocol packet may be stored in the broker-side receive buffer.

[0045] The broker may acknowledge receipt of the packets by sending one or more acknowledgement (ACK) packets to the client via the tunnel connection. The broker may generate and send one or more acknowledgement (ACK) packets to the Web server. In one embodiment, the broker may store ACK packets in a broker-side transmit buffer. In one embodiment, the ACK packets may be sent to the Web server over the TCP connection. In one embodiment, the broker-side transport protocol tunnel driver may handle the transmission of the ACK packets to the Web server.

[0046] The Web server may then receive the ACK packets. In one embodiment, a transport protocol tunnel servlet may receive the packets for the Web server. The Web server may store the acknowledgement packets in a transport protocol packet buffer. At some point, the client may send a transport protocol pull request packet to the Web server. In one embodiment, the client may periodically send pull requests to the Web server. In one embodiment, a separate thread on the client may handle periodically sending pull requests. The Web server may transmit to the client the one or more transport protocol ACK packets stored in the transport protocol packet buffer associated with the client in response

to receiving the pull request packet. The client may then receive the one or more ACK packets. The ACK packets may serve to acknowledge the receipt of the transmitted data so that the sender may free its transmit buffers.

[0047] Using embodiments of the transport protocol tunnel connection layer, messaging system messages may be generated on a broker and sent to a client. In one embodiment, the generated messages may be stored in a broker-side transmit buffer. One or more transport protocol packets encapsulating the one or more messages may then be generated on the broker. In one embodiment, a broker-side transport protocol tunnel connection driver may generate the transport protocol packets and include the messages as payloads of the packets. The one or more transport protocol packets may then be transmitted to the client via the transport protocol tunnel connection. In one embodiment, the broker-side transport protocol tunnel connection driver may handle the transmission of the packets. In one embodiment, the transport protocol packets may be sent to a Web server. In one embodiment, the transport protocol packets may be sent to the Web server over a TCP connection.

[0048] The Web server may receive the transport protocol packets and store the received packets in a transport protocol packet buffer. In one embodiment, a transport protocol tunnel servlet may receive the packets for the Web server. At some point, the client may send a transport protocol pull request packet to the Web server. The Web server may transmit to the client one or more transport protocol packets stored in the transport protocol packet buffer associated with the client in response to receiving the pull request packet.

[0049] The client may then receive the one or more transport protocol packets. In one embodiment, the client may store the received transport protocol packets in a client-side receive buffer. After receiving the transport protocol packets, the client may acknowledge receipt of the packets by sending one or more acknowledgement (ACK) packets to the broker via the transport protocol tunnel connection. The Web server may receive the ACK packets and forward the received ACK packets to the broker. In one embodiment, the ACK packets may be transmitted to the broker via a TCP connection between the Web server and the broker that serves as one segment of the HTTP tunnel connection. The broker may receive the ACK packets from the Web server. In one embodiment, the received ACK packets may be stored in a broker-side receive buffer.

[0050] In one embodiment, one Web server and one tunnel servlet may be used by two or more clients to communicate to a broker via tunnel connections. In this embodiment, the HTTP tunnel servlet may multiplex transport protocol packets from the two or more clients onto the TCP connection. In one embodiment, the tunnel servlet may extract the messaging system message information from received transport protocol packets and send only the message information to the broker over the TCP connection. In one embodiment, there may be one broker-side receive buffer for each tunnel connection. In another embodiment, a single receive buffer may be used for two or more tunnel connections.

[0051] Transport protocol packets may optionally pass through a proxy server and one or more firewalls. For example, transport protocol packet flow from a client to a broker may pass through a proxy server, through a firewall onto the Internet, through another firewall to a Web server,

and from the Web server over a TCP connection to the broker. Packets flowing in the opposite direction (from the broker to the client) take the reverse path.

[0052] In one embodiment, transport protocol packets transmitted on the tunnel connection may include message sequence information configured for use by the receiver in processing received messages in the correct sequence. In one embodiment, flow control may be applied to the sending of messages from a sender to a receiver. In one embodiment, the receiver may inform the sender of available space in a receive buffer to store incoming messages. In one embodiment, upon establishment of a tunnel connection, each side (both clients and brokers can be senders and/or receivers) may inform the other of its receive buffer size. In one embodiment, ACK packets sent to a sender by a receiver may each include the currently available space in the receive buffer. Thus, the sender can keep track of the current receive capacity of the receiver.

BRIEF DESCRIPTION OF THE DRAWINGS

[0053] FIG. 1 illustrates a prior art messaging-based application based upon client/server architecture;

[0054] FIG. 2 illustrates a prior art messaging-based application based on point-to-point architecture;

[0055] FIG. 3A illustrates a client-server messaging system implementing a transport protocol tunnel connection layer according to one embodiment;

[0056] FIG. 3B illustrates a client-server messaging system implementing a transport protocol tunnel connection layer with multiple clients accessing a broker through a Web server according to one embodiment;

[0057] FIG. 4 illustrates the architecture of a client-server messaging system implementing a transport protocol tunnel connection layer according to one embodiment;

[0058] FIG. 5A illustrates the routing of transport protocol packets between clients and a broker on transport protocol tunnel connections according to one embodiment;

[0059] FIG. 5B illustrates the process of sending a message from a client to a broker via a transport protocol tunnel connection according to one embodiment;

[0060] FIG. 5C illustrates the process of sending a message from a broker to a client via a transport protocol tunnel connection according to one embodiment;

[0061] FIGS. 6A is a flowchart of a method for sending messages from a messaging system client to a messaging system broker over a transport protocol tunnel connection layer according to one embodiment; FIG. 6B is a flowchart of a method for transmitting one or more packets from a client to a broker via a transport protocol tunnel connection according to one embodiment;

[0062] FIG. 6C is a flowchart of a method for a broker to acknowledge to a client the receipt of transport protocol packets via the transport protocol tunnel connection according to one embodiment;

[0063] FIG. 7A is a flowchart of a method for sending messages from a messaging system client to a messaging system broker over a transport protocol tunnel connection layer according to one embodiment;

[0064] FIG. 7B is a flowchart of a method for transmitting one or more packets from a broker to a client via a transport protocol tunnel connection according to one embodiment;

[0065] FIG. 7C is a flowchart of a method for a client to acknowledge to a broker the receipt of transport protocol packets via a transport protocol tunnel connection according to one embodiment; and

[0066] FIG. 8 illustrates an exemplary transport protocol packet format that may be used in the transport protocol tunnel connection layer according to one embodiment.

[0067] While the invention is described herein by way of example for several embodiments and illustrative drawings, those skilled in the art will recognize that the invention is not limited to the embodiments or drawings described. It should be understood, that the drawings and detailed description thereto are not intended to limit the invention to the particular form disclosed, but on the contrary, the intention is to cover all modifications, equivalents and alternatives falling within the spirit and scope of the present invention as defined by the appended claims. The headings used herein are for organizational purposes only and are not meant to be used to limit the scope of the description or the claims. As used throughout this application, the word "may" is used in a permissive sense (i.e., meaning having the potential to), rather than the mandatory sense (i.e., meaning must). Similarly, the words "include", "including", and "includes" mean including, but not limited to.

DETAILED DESCRIPTION OF EMBODIMENTS OF THE INVENTION

[0068] A system and method for providing transport protocol tunnel connections between entities such as clients and servers in a messaging system is described. A transport protocol tunnel connection layer is described that may be used to provide reliable, full duplex virtual connections between entities (e.g. clients and servers) in a distributed application environment using a messaging system. This layer may be used by clients to access messaging servers (referred to as brokers) through proxy servers and firewalls, thus expanding the scope of from where clients can access brokers. Using this layer, brokers as well as clients may initiate messages in the distributed application environment. This layer may also provide guaranteed data delivery with correct sequencing even in case of a failure on the network. This layer may also provide end-to-end flow control. The transport protocol tunnel connection layer may be used to simulate virtual connections that have a similar contract with the upper layers (e.g. clients and brokers) as a TCP connection. Thus, applications can be developed in terms of messages without concern for the particular underlying connection protocol. Decisions about the best communications protocol between clients and brokers may be postponed until deployment time when the particular network requirements of an installation are known.

[0069] The transport protocol tunnel connection layer may use a novel tunneling protocol to overcome limitations of the underlying transport protocol in the messaging environment. A transport protocol may be designed as a stateless request-response mechanism, and thus may not fit the enterprise messaging system protocol model very well. For example, enterprise messaging system applications may expect asynchronous message delivery whereas a transport protocol

may use a synchronous request response model. As another example, each transport protocol request-response exchange may carry a finite amount of data and is usually short lived. Thus, an enterprise messaging system message may be delivered using multiple transport protocol requests. However there may be no correlation between transport protocol requests generated by the same client. The Web servers and transport protocol intermediaries (e.g. proxy servers) thus cannot guarantee that the transport protocol requests will be processed in the same sequence as they were sent. As yet another example, TCP protocol uses a flow control mechanism to limit the network resource usage. If an enterprise messaging system message sender application generates messages very rapidly, and if each message is sent to the server using a separate transport protocol request or multiple transport protocol requests, resources on the transport protocol servers and intermediaries may be exhausted. As still yet another example, an enterprise messaging system client application may maintain a connection and a steady message exchange rate over very long periods of time (many days or even months). If the transport protocol is used, a failure on a common transport protocol proxy server may disrupt the communication between the client application and the enterprise messaging system broker.

[0070] The transport protocol tunnel connection layer may provide a connection-oriented, bi-directional byte stream service between nodes in a messaging system. Transport protocol messages may be carried as transport protocol packet payloads. The transport protocol tunnel connection layer may allow messaging system clients to access messaging system brokers using the transport protocol instead of using direct TCP connections. This enables the clients to access the brokers through firewalls. Also, with the help of a transport protocol proxy, the clients may access the messaging service even when there is no direct IP connectivity with the broker.

[0071] Using the transport protocol tunnel connection layer, if a client is separated from a broker by a firewall, messaging may be run on top of transport protocol connections, which are normally allowed through firewalls. On the client side, a transport protocol transport driver may encapsulate messages into transport protocol packets, and also may ensure that these packets are sent to the Web server in the correct sequence. The client may use a transport protocol proxy server to communicate with the broker if necessary. In one embodiment, a transport protocol tunnel servlet executing within a Web server may receive the transport protocol packets and forward the entire transport protocol packets (including the message data) to the broker. In another embodiment, the transport protocol tunnel servlet executing within the Web server may be used to pull client data (messages) out of the transport protocol packets before forwarding the data to the broker. In one embodiment, the tunnel servlet may multiplex message data from multiple clients onto one TCP connection to the broker, thus allowing the use of one tunnel servlet for multiple clients. The transport protocol tunnel servlet may also send broker data (messages) to the client in response to transport protocol pull requests. On the broker side, a transport protocol transport driver may unwrap and demultiplex incoming messages from the transport protocol tunnel servlet. The transport protocol tunnel connection layer may also be used by brokers to initiate communications with a client.

[0072] Though embodiments of the system and method are described herein as providing Hypertext Transport Protocol (HTTP) tunnel connections between entities such as clients and servers in a messaging system, it is noted that embodiments of the system and method using other unreliable/connectionless transport protocols that may not have built-in TCP support such as UDP (User Datagram Protocol), IrDA (Infrared Data Association), IBM's SNA (Systems Network Architecture), Novell's IPX (Internetwork Packet eXchange), and Bluetooth are contemplated. These embodiments may be used to provide tunnel connections between entities in messaging systems using the other transport protocols.

[0073] FIG. 3A illustrates a client-server messaging system implementing an HTTP tunnel connection layer according to one embodiment. Client 200 may generate messages using the messaging protocol 212. In one embodiment, an Application Programming Interface (API) to the messaging protocol may be used by the client application to generate the messages. In one embodiment, the messaging protocol is the Java Message Service (JMS). Other messaging protocols may be used. Generated messages may then be passed to the HTTP tunnel client driver 220.

[0074] The HTTP tunnel client driver 220 may then send the messages as HTTP POST request payloads. The HTTP tunnel client driver 220 may also use separate HTTP requests to periodically pull any data sent by the other end of the connection. The HTTP requests may be sent through HTTP proxy 206, Internet 204, and firewall 210 to Web server 208. On Web server 208, the HTTP tunnel servlet 214 may act as a transceiver and may multiplex the HTTP requests from multiple clients onto a single TCP connection 216 with the broker 202. The HTTP tunnel broker driver 240 may receive the HTTP requests from the Web server 210 over TCP connection 216.

[0075] Note that the HTTP proxy 206 and/or the firewall 210 are optional. In other words, the HTTP tunnel connection layer is configurable to transmit messages encapsulated in HTTP requests between entities over the Internet 104 both with and without the messages passing through proxies and/or firewalls. Also note that the Web server 208, HTTP tunnel servlet 214, and the broker 202 may be implemented on the same host machine or on different host machines. Note also that a Web server 208 and HTTP tunnel servlet 214 may be used to access multiple brokers 202.

[0076] In one embodiment, the packet delivery service provided by the HTTP tunnel drivers may occasionally lose packets. Hence the HTTP tunnel connection layer may use positive acknowledgements of received packets, and may retransmit any lost packets. When a receiver receives a packet including a message, the receiver responds to the packet by sending an acknowledgement packet to the sender.

[0077] In one embodiment, the HTTP tunnel connection layer may use a sliding window protocol to implement packet sequencing and flow control on top of HTTP. In a distributed application environment using a messaging service, quite often entities may need to send a stream of packets, without errors, and with guaranteed order of delivery (i.e. that the packets are received in the order they are sent). HTTP does not guarantee packets to be received by a receiver (e.g. broker) in the same order the packets were sent by a transmitter (e.g. client). Also, HTTP does not provide

a method to control the rate at which packets are sent to a receiver, thus running the risk of exhausting network resources on the receiver and losing packets. Using a sliding window protocol provides the ability to control the rate at which packets are transmitted to a receiver. The sliding window protocol may be used to guarantee that no more than a fixed number of packets are transmitted to a receiver. The fixed number of packets may be determined by a receive buffer size on the receiver. For example, a receiver may be able to receive a maximum of 100 packets from a sender. If the sender initially transmits 70 packets, the receiver may receive the packets and process 20 of them. The receiver may send a packet or packets to notify the sender that the receiver can receive 50 packets. The sender may then transmit 30 more packets. The receiver may receive the 30 packets, and in the meantime may have processed 20 more of the original 70 packets. The receiver may then transmit a packet or packets to notify the sender that the receiver can receive 60 packets (there are still 10 of the original 70 packets and the 30 new packets in the receive buffer). Thus, the sender never sends more packets to the receiver than the quantity of packets the receiver has notified the sender it can receive.

[0078] In one embodiment, when a connection is established between two entities or nodes (e.g. a server and client), each node may inform the other of how many packets it can initially receive. In a network, a node is a connection point, either a redistribution point or an end point for data transmissions. In general, a node has programmed or engineered capability to recognize, process and/or forward transmissions to other nodes. In one embodiment, each packet received by the receiver may be acknowledged with a packet sent to the sender. The acknowledgement packets may each include information indicating the current receive buffer size (i.e. the number of packets that the receiver can currently receive). Thus, the sender can determine the number of packets that it can send to the receiver without overwhelming the sender.

[0079] In one embodiment, the client 200, broker 202 and the messaging protocol layers 212 may function similarly whether the underlying transport protocol is TCP or HTTP. In one embodiment, the messaging protocol layers 212 on both the client and the broker may use the same basic design and threading model for TCP and HTTP support. In one embodiment, an enterprise messaging system using the HTTP tunneling protocol may allow a messaging system application to exchange messages using the TCP, HTTP, SSL, or other protocol by changing appropriate configuration parameters at runtime. Thus, the application developer may not have to write any transport specific code. A client library and the broker 202 may handle the transport-specific details.

[0080] FIG. 3B illustrates a client-server messaging system implementing the HTTP tunnel connection layer with multiple clients accessing a broker through a Web server according to one embodiment. Both brokers 202A and 202B may have registered with Web server 208 that they are ready to receive connections from clients. Client 200A may establish an HTTP tunnel connection to broker 202A through Web server 208. Client 200B may establish an HTTP tunnel connection to broker 202B through Web server 208. At some point, either client may establish an HTTP tunnel connection with the other broker. Thus, a client may have multiple

HTTP tunnel connections open to different brokers, and multiple clients may have HTTP tunnel connections open to one broker.

[0081] FIG. 4 illustrates the architecture of a client-server messaging system implementing the HTTP tunnel connection layer according to one embodiment. Various components that are comprised in the network protocol stack for the HTTP tunnel connection layer are shown. These components lie between the application (client or broker) and the HTTP tunnel driver (client or broker). These components may include an HTTP tunnel input stream/HTTP tunnel output stream 270, an HTTP tunnel socket 272, and an HTTP tunnel connection 274. In one embodiment, these components may be implemented as classes. In one embodiment, these components may be implemented as Java classes.

[0082] The HTTP tunnel drivers may send one request or receive one packet at a time. The primary responsibility of these drivers is to make sure that single packets are sent and/or received. This driver may not be aware of or be involved directly in the ordering, flow control or acknowledgement of packets.

[0083] The HTTP tunnel connection component 274 may interpret received packets. This component may be responsible for the sliding window protocol implementation. This component may be responsible for buffering the packets, may implement flow control, reordering and other aspects of the HTTP tunneling protocol. This component may be the primary component that is used to implement the end-to-end HTTP tunnel "virtual" connection as described herein.

[0084] The HTTP tunnel socket 272 and input and output streams 270 provide an interface between the application (e.g. client or broker) and the HTTP tunnel connection 274. These components may include simple, atomic methods such as read, write and open connection methods that provide an abstract interface to the HTTP tunnel connection 274 for the applications, and thus may hide the HTTP tunnel connection implementation from the applications.

[0085] HTTP tunnel server socket 276 may be used by broker 202 to open a listening socket to the Web server 208 to allow the Web server to create a listening endpoint for the broker 202. In doing so, broker 202 is establishing that it is ready to accept HTTP tunnel connections through Web server 208. Thus, when client 200 desires to open an HTTP tunnel connection to broker 202, the client may contact Web server 208 with a packet including information identifying broker 202. Web server 208 may examine the packet and forward the packet to broker 202, where an HTTP tunnel connection to client 200 may be established. Note that one or more HTTP packet buffers may be allocated on Web server 208 for the newly established HTTP tunnel connection. Thus, a client 200 initiates a connection, and a broker 202 accepts the connection. In one embodiment, brokers 202 cannot initiate connections to clients. Therefore, once an HTTP tunnel connection between a client 200 and a broker 202 is open (initiated by the client 200), the client 200 may keep the connection open so that if the broker 202 has data to send to the client, there is a communications link established for the broker to send messages to the client on. In one embodiment, the broker 202 never sends HTTP packets directly to the client 200; the packets are buffered on Web server 208, and pulled from the Web server periodically by the client 200.

[0086] FIGS. 5A through 5C are data flow diagrams illustrating the operation of a client-broker messaging system implementing the HTTP tunnel connection layer according to one embodiment. FIG. 5A illustrates the routing of HTTP packets between clients 200 and a broker 202 on HTTP tunnel connections 292 according to one embodiment. Each client 200 may include at least one client-side transmit buffer 222 and at least one client-side receive buffer 224. Broker 202 may include at least one broker-side receive buffer 244 and at least one broker-side transmit buffer 242. Each client 200 may establish an HTTP tunnel connection 292 to broker 202, which may pass through network 294, through Web server 208, and over a TCP connection 296 to broker 202. There may be a single TCP connection 296 between Web server 208, or alternatively a TCP connection 296 may be established for each HTTP tunnel connection 292. In passing through network 294, an HTTP tunnel connection 292 may pass through a proxy server (not shown) and/or through one or more firewalls (not shown).

[0087] Clients 200 may generate messaging system messages, which may be buffered in a client-side transmit buffer 222. Buffering message data may allow messages to be retransmitted if necessary, for example. The clients 200 may generate HTTP packets that include the message data as payloads and transmit the HTTP message packets to broker 202 via the HTTP tunnel connections 292 as indicated at 280A and 280B. Web server 208 may receive the HTTP message packets from network 294 and forward the packets to broker 202 over a TCP connection 296.

[0088] Broker 202 may buffer incoming messages in a broker-side receive buffer 244. Broker 202 may generate an acknowledgment (ACK) HTTP packet to acknowledge the receipt of each HTTP packet successfully received from a client 200, and send the ACK packets to Web server 208 over a TCP connection 296 as indicated at 282A and 282B. Web server 208 may buffer the ACK packets received from broker 202 in buffers 250. In one embodiment, there may be a buffer 250 for each HTTP tunnel connection 292; in other words, each connection 292 may use a separate instance of buffer 250. In another embodiment, one buffer 250 may be shared among two or more HTTP tunnel connections 292.

[0089] Using the HTTP tunneling protocol layer, a broker 202 as well as clients 200 may initiate messaging system messages. Messages generated by broker 202 may be buffered in a broker-side transmit buffer 242. Buffering message data may allow messages to be retransmitted if necessary, for example. The broker 202 may generate HTTP packets that include the message data as payloads and transmit the HTTP packets to Web server 208 over a TCP connection 296 as indicated at 288A and 288B. Web server 208 may buffer the HTTP message packets received from broker 202 in buffers 250.

[0090] As indicated at 284A and 284B, each client 200 may send an HTTP request packet to Web server 208 to indicate that the client 200 is ready to receive HTTP packets buffered in a buffer 250 for the client 200. In one embodiment, each client may periodically send HTTP request packets to retrieve buffered HTTP packets from Web server 208. In one embodiment, a separate thread on a client 200 may be responsible for periodically sending the HTTP request packets.

[0091] After Web server 208 receives an HTTP request packet from a client 200 as indicated at 284, the Web server

208 may respond by sending the requesting client 200 one or more HTTP packets currently buffered in a buffer 250 for the client 200. The HTTP packets may include ACK packets as sent at 282 and/or HTTP message packets as sent at 288. Upon receiving HTTP packets from the Web server 208, a client 200 may store the received packets in a client-side receive buffer 224 to await processing. A client 200 may also generate an ACK packet and send them to broker 202 over the HTTP tunnel connection, as indicated at 290A and 290B, in response to each HTTP message packet received.

[0092] FIG. 5B illustrates the process of sending a message from a client 200 to a broker 202 via an HTTP tunnel connection according to one embodiment. At 300, the client 200 may generate a message 260A. At 302, the client side HTTP tunneling driver 220 may receive the message 260A and buffer the outgoing message data in a transmit buffer 222, which may allow the message data to be retransmitted if necessary. The client side HTTP tunneling driver 220 generates an HTTP POST request with the message data as payload as indicated at 304. This HTTP request may travel over the Internet. The client 200 may be configured to send HTTP requests via a proxy server 206 and through one or more firewalls 210 if necessary.

[0093] A Web server 208 may receive the HTTP request with the message data, and forward the HTTP request to the server side HTTP tunneling driver 240 over a TCP connection as indicated at 306. The server side HTTP tunneling driver 240 may store the received message data in a receive buffer 244. The packet header of the HTTP request may include a sequence number for use in preserving message order when multiple messages are transmitted. The message data may remain in receive buffer 244 until the broker 202 consumes it.

[0094] The server side HTTP tunneling driver may generate an acknowledgement (ACK) HTTP packet to indicate successful receipt of the HTTP request packet and message data. The ACK packet may include information about how much space is left in the receive buffer 244. This information may be used as a "flow control" mechanism to slow down the sender (client) if the receiver (broker) cannot consume the data fast enough. The ACK packet may be sent to the Web server 208 over the TCP connection as indicated at 308. The ACK packet may be stored in packet buffer 250A on the Web server 208 waiting for an HTTP request from the client 200. In one embodiment, the Web server 208 may not initiate communication with the client 200; it can only respond to incoming HTTP requests.

[0095] The client side HTTP tunneling driver 220 then may send an HTTP request packet to the Web server 208 to pull any pending HTTP packets as indicated at 310. In one embodiment, the client side HTTP tunneling driver 220 may use a separate reader thread that continuously sends requests to the Web server 208 to pull any pending HTTP packets. After the Web server 208 receives a pull request for the HTTP packet buffered at 308, the Web server may send the buffered HTTP packet(s) to client 200 in response to the pull request as indicated at 312. The client side HTTP tunneling driver 220 then may process the HTTP packet(s) including the ACK packet, and free the corresponding message data buffered in transmit buffer 222 at 302. In one embodiment, information from the HTTP packet(s) sent to the client 200 may be stored in a client-side receive buffer (not shown) and accessed from the client-side receive buffer for processing.

[0096] FIG. 5C illustrates the process of sending a message from the broker 202 to the client 200 via an HTTP tunnel connection according to one embodiment. At 400, the broker 202 generates a message 260B. At 402, the broker side HTTP tunneling driver 240 may receive the message 260B and buffer the outgoing message data in a transmit buffer 242 so that it can be retransmitted if necessary. The broker side HTTP tunneling driver 240 may generate an HTTP packet with the message data as payload and forward it to Web server 208 over a TCP connection as indicated at 404. The Web server 208 receives the HTTP packet with the message data, and writes the packet to buffer 250. The server side HTTP tunneling driver 240 may send an HTTP request to the Web server 208 to pull any pending packets as indicated at 406. When the Web server 208 receives the pull request, it sends the packet buffered at 404 in response to the pull request as indicated at 408. The client side HTTP tunneling driver 220 may store the received message data in a receive buffer 224. The packet header of the HTTP request may include a sequence number to preserve message order. The message data may remain in receive buffer 224 until the client 200 consumes it.

[0097] The client side HTTP tunneling driver generates an acknowledgement (ACK) HTTP packet to indicate successful receipt of the HTTP request and message data. The ACK packet may also include information about how much space is left in receive buffer 224. This information may be used as a "flow control" mechanism to slow down the sender (broker) if the receiver (client) cannot consume the data fast enough. The ACK packet may be sent to the Web server 208 over the Internet as indicated at 410. Web server 208 may forward the ACK packet to the server side HTTP tunneling driver 240 over a TCP connection as indicated at 412. The server side HTTP tunneling driver 240 then may free the corresponding message data packet(s) buffered in transmit buffer 242 at 402. In one embodiment, information from the ACK packet sent to the broker 202 may be stored in a broker-side receive buffer (not shown) and accessed from the broker-side receive buffer for processing.

[0098] FIGS. 6A-6C are flowcharts illustrating a method of sending messages from a messaging system client to a messaging system broker over an HTTP tunnel connection layer according to one embodiment. In FIG. 6A, a transport protocol tunnel connection between a client and a broker in a messaging system may be established as indicated at 600. As indicated at 602, one or more messaging system messages may be generated on the client. In one embodiment, the generated messages may then be stored in a client-side transmit buffer. One or more transport protocol packets encapsulating the one or more messages may be generated on the client as indicated at 604. In one embodiment, a client-side HTTP tunnel connection driver may generate the HTTP packets and include the messages as payloads of the packets. As indicated at 606, the one or more HTTP packets may then be transmitted to the broker via the HTTP tunnel connection. In one embodiment, the client-side HTTP tunnel connection driver may handle the transmission of the packets.

[0099] In one embodiment, each HTTP packet transmitted on the HTTP tunnel connection may include message sequence information configured for use by the receiver in processing received messages in the correct sequence. This is useful since HTTP and some other transport protocols do

not guarantee delivery of messages in order. In one embodiment, flow control may be applied to the sending of messages from a sender to a receiver. In flow control, before sending new messages, the sender may determine available resources on the receiver to receive new messages, and then transmit no more messages than the receiver can handle based upon the available resources. In one embodiment, the receiver may inform the sender of available space in a receive buffer to store incoming messages awaiting processing. In one embodiment, upon establishment of an HTTP tunnel connection, each side (both clients and brokers can be senders and/or receivers) may inform the other of its receive buffer size.

[0100] As indicated at 608, the broker may receive the HTTP packets and then may acknowledge receipt of the one or more packets by sending one or more acknowledgement (ACK) packets to the client via the HTTP tunnel connection. In one embodiment, the ACK packets may each include the currently available space in the broker-side receive buffer. Thus, the sender (client) can keep track of the current receive capacity of the receiver (broker).

[0101] FIG. 6B expands on 606 of FIG. 6A and illustrates a method of transmitting one or more packets from a client to a broker via an HTTP tunnel connection according to one embodiment. As indicated at 620, the transmitted HTTP packets may optionally pass through a proxy server. As indicated at 622, the transmitted HTTP packets may optionally pass through one or more firewalls. A Web server may then receive the one or more HTTP packets as indicated at 624. For example, the client may transmit the packets to a proxy server, the proxy server may transmit the packets through a firewall onto the Internet, and the packets may be received through another firewall by the Web server.

[0102] The Web server may then forward the received HTTP packets to the broker as indicated at 626. In one embodiment, the packets may be transmitted to the broker via a TCP connection between the Web server and the broker that serves as one segment of the HTTP tunnel connection. In one embodiment, an HTTP tunnel servlet on the Web server node may serve as an interface between the Web server and the TCP connection to the broker. In one embodiment, one Web server and HTTP tunnel servlet may be used by two or more clients to communicate to a broker via HTTP tunnel connections. In this embodiment, the HTTP tunnel servlet may multiplex HTTP packets from the two or more clients onto the TCP connection. In one embodiment, the HTTP tunnel servlet may extract the messaging system message information from received HTTP packets and send only the message information to the broker over the TCP connection.

[0103] As indicated at 628, the broker may receive the HTTP packets from the Web server. In one embodiment, the broker may receive the packets on a TCP connection to the Web server. In one embodiment, a broker-side HTTP tunnel driver may receive the packets. In one embodiment, the broker may extract the messaging system messages from the HTTP packets and store the messages in a broker-side receive buffer. In another embodiment, the entire HTTP packet may be stored in a receive buffer. In one embodiment, there may be one receive buffer on the broker for each HTTP tunnel connection. In another embodiment, a single receive buffer may be used for two or more HTTP tunnel connections.

[0104] FIG. 6C expands on 608 of FIG. 6A and illustrates a method of a broker acknowledging to a client the receipt of HTTP packets from the client via the HTTP tunnel connection according to one embodiment. As indicated at 640, the broker may generate and send one or more acknowledgement (ACK) packets to the Web server. In one embodiment, the broker may store ACK packets in a broker-side transmit buffer. In one embodiment, the broker may send one ACK packet for each received HTTP packet. In another embodiment, the broker may send one ACK packet for each completely received messaging system message. In one embodiment, the ACK packets may be sent to the Web server over a TCP connection. In one embodiment, a broker-side HTTP tunnel driver may handle the transmission of the ACK packets on the TCP connection. In one embodiment, an HTTP tunnel servlet may receive the packets for the Web server.

[0105] As indicated at 642, the Web server may store the acknowledgement packets in an HTTP packet buffer. In one embodiment, there may be one HTTP packet buffer for each HTTP tunnel connection supported by the Web server. In another embodiment, there may be two HTTP packet buffers per tunnel connection, with one HTTP packet buffer for each message flow direction on each tunnel connection. In yet another embodiment, one or more HTTP packet buffers may be used for two or more tunnel connections.

[0106] As indicated at 644, at some point, the client may send an HTTP pull request packet to the Web server. In one embodiment, the client may periodically send HTTP pull requests to the Web server. In one embodiment, a separate thread on the client may handle periodically sending pull requests. The Web server may transmit to the client the one or more HTTP ACK packets stored in the HTTP packet buffer associated with the client in response to receiving the pull request packet as indicated at 646. As indicated at 648, the transmitted ACK packets may optionally pass through one or more firewalls. As indicated at 750, the transmitted ACK packets may optionally pass through a proxy server. As indicated at 652, the client may then receive the one or more ACK packets. In one embodiment, each ACK packet may include information on available space in the broker-side receive buffer to be used in flow control of HTTP packets from the client to the broker. The ACK packets may serve to acknowledge the receipt of the transmitted data so that the sender may free its transmit buffers.

[0107] FIGS. 7A-7C are flowcharts illustrating a method of sending messages from a messaging system client to a messaging system broker over an HTTP tunnel connection layer according to one embodiment. In FIG. 7A, a transport protocol tunnel connection between a client and a broker in a messaging system may be established as indicated at 700. As indicated at 702, one or more messaging system messages may be generated on the broker. In one embodiment, the generated messages may then be stored in a broker-side transmit buffer. One or more transport protocol packets encapsulating the one or more messages may be generated on the broker as indicated at 704. In one embodiment, a broker-side HTTP tunnel connection driver may generate the HTTP packets and include the messages as payloads of the packets. As indicated at 706, the one or more HTTP packets may then be transmitted to the client via the HTTP tunnel

connection. In one embodiment, the broker-side HTTP tunnel connection driver may handle the transmission of the packets.

[0108] As indicated at 708, the client may receive the HTTP packets, and then may acknowledge receipt of the one or more packets by sending one or more acknowledgement (ACK) packets to the broker via the HTTP tunnel connection. In one embodiment, the ACK packets may each include the currently available space in the client-side receive buffer. Thus, the sender (broker) can keep track of the current receive capacity of the receiver (client).

[0109] FIG. 7B expands on 706 of FIG. 7A and illustrates a method of transmitting one or more packets from a broker to a client via an HTTP tunnel connection according to one embodiment. As indicated at 720, the broker may generate and send one or more HTTP packets to the Web server. In one embodiment, the broker may store the HTTP packets in a broker-side transmit buffer. In one embodiment, the HTTP packets may be sent to the Web server over a TCP connection. In one embodiment, a broker-side HTTP tunnel driver may handle the transmission of the HTTP packets on the TCP connection.

[0110] As indicated at 722, the Web server may receive the HTTP packets and store the received HTTP packets in an HTTP packet buffer. In one embodiment, an HTTP tunnel servlet may receive the packets for the Web server. As indicated at 724, at some point, the client may send an HTTP pull request packet to the Web server. In one embodiment, the client may periodically send HTTP pull requests to the Web server. The Web server may transmit to the client the one or more HTTP packets stored in the HTTP packet buffer associated with the client in response to receiving the pull request packet as indicated at 726. As indicated at 728, the transmitted HTTP packets may optionally pass through one or more firewalls. As indicated at 730, the transmitted HTTP packets may optionally pass through a proxy server. As indicated at 732, the client may then receive the one or more HTTP packets. In one embodiment, the client may store the received HTTP packets in a client-side receive buffer. In one embodiment, each HTTP packet may include sequence information configured for use by the client in processing received messages in sequence.

[0111] FIG. 7C expands on 708 of FIG. 7A and illustrates a method of a client acknowledging to a broker the receipt of HTTP packets from the broker via the HTTP tunnel connection according to one embodiment. As indicated at 740, the client may generate and send one or more acknowledgement (ACK) packets to the broker. As indicated at 742, the transmitted ACK packets may optionally pass through a proxy server. As indicated at 744, the transmitted ACK packets may optionally pass through one or more firewalls. A Web server may then receive the one or more ACK packets as indicated at 746. For example, the client may transmit the ACK packets to a proxy server, the proxy server may transmit the ACK packets through a firewall onto the Internet, and the ACK packets may be received through another firewall by the Web server.

[0112] The Web server may then forward the received ACK packets to the broker as indicated at 748. In one embodiment, the ACK packets may be transmitted to the broker via a TCP connection between the Web server and the broker that serves as one segment of the HTTP tunnel

connection. In one embodiment, an HTTP tunnel servlet on the Web server node may serve as an interface between the Web server and the TCP connection to the broker.

[0113] As indicated at 750, the broker may receive the ACK packets from the Web server. In one embodiment, the broker may receive the packets on a TCP connection to the Web server. In one embodiment, a broker-side HTTP tunnel driver may receive the ACK packets. In one embodiment, the received ACK packets may be stored in a broker-side receive buffer. In one embodiment, each ACK packet may include information on available space in the client-side receive buffer to be used in flow control of HTTP packets from the broker to the client. The methods as described in FIGS. 6A-6C and FIGS. 7A-7C may be implemented in software, hardware, or a combination thereof. The order of method may be changed, and various steps may be added, reordered, combined, omitted, modified, etc.

[0114] FIG. 8 illustrates an exemplary HTTP tunneling protocol packet format according to one embodiment. The HTTP tunnel drivers at either end of a connection may encode all their data packets using this packet format. Each messaging system message may be carried separately as a HTTP request or response payload. A messaging system message may be sent as the payload of a single packet or, alternatively, the message may be broken into parts and sent as the payload of two or more packets.

[0115] The packet format may include several fields. Version 500 may indicate a version number of the packet that may be used to indicate different releases of the HTTP tunnel connection layer software. In one embodiment, the version 500 may be a 16-bit field. Packet type 502 may indicate the type of the packet. This field may be used to indicate to the HTTP tunnel driver at the other end what to do with the contents of this packet. In one embodiment, the packet type field may be 16 bits. Size 504 may indicate the size of the entire packet including the header. In one embodiment, this may be a 32-bit field. Connection ID 506 may be a unique integer that may be used as connection identifier. This value may be assigned at the time of connection establishment. This field may be used by the server to distinguish between connections to multiple clients. In one embodiment, this may be a 32-bit field. Packet sequence number 508 may be used to ensure sequential delivery of data bytes and flow control. In one embodiment, each packet may be assigned a unique incremental sequence number by the sender. In one embodiment, this may be a 32-bit field. One embodiment may include a receiver window size 510 that may be used for flow control and may indicate the capacity of the receive side buffer. In one embodiment, this may be a 16-bit field. One embodiment may guarantee that the packets are either delivered correctly, or in case of an error, are not delivered at all. Some embodiments may not use checksum 512, and thus this field may be reserved.

[0116] One embodiment of the HTTP tunnel connection layer may use a connection establishment protocol that may be used to initialize the following connection state components:

[0117] The HTTP tunnel servlet 214 allocates a unique connection ID for the new connection. It also may allocate one or more buffers 250 for packet streams in both directions.

[0118] The HTTP tunnel drivers on both server and client side allocate and initialize the transmit buffers and receive buffers for packet streams in both directions.

[0119] The following are examples of HTTP tunneling protocol packets that may be used in the connection establishment protocol and are not intended to be limiting. Note that one skilled in the art will recognize that other packet formats may be used within the scope of the invention.

[0120] A client 200 may initiate a connection to a server 202 by sending the following information in an HTTP packet to the HTTP tunnel servlet 214:

[0121] URL parameters="?ServerName=<ServerIdString>&Type=connect" Packet Type=connection initialization packet (e.g. CONN_INIT_PACKET (1)).

[0122] Connection ID =0

[0123] The servlet 214 may allocate a unique connection ID for this connection and send it to the client 200 and the server 202 in HTTP packets including the following information:

[0124] Packet Type=connection initialization packet (e.g. CONN_INIT_PACKET (1)).

[0125] ConnectionID=<connid>

[0126] This information may be sent to the client 200 as a response payload for the HTTP request that carried the client's connection initialization packet.

[0127] The server 202 may acknowledge the connection initialization packet to the client 200 in an ACK packet that may include the following information:

[0128] Pull URL parameters=

[0129] "?ServerName=<ServerIdString>&Type=pull&ConnId=<connid>"

[0130] Packet Type=

[0131] connection initialization acknowledgement (e.g. CONN_INIT_ACK (2)).

[0132] After completion, both client 200 and server 202 are aware of the newly established connection and normal data exchange can begin.

[0133] Once the connection is established, both sides may start sending data and connection management packets. The following are examples of HTTP tunneling protocol packets that may be used as data and connection management packets and are not intended to be limiting. Note that one skilled in the art will recognize that other packet formats may be used within the scope of the invention.

[0134] From the client 200 to the HTTP tunnel servlet 214 to the server 202:

[0135] URL parameters="?ServerName=<ServerIdString>&Type=push"

[0136] From the server 202 to the HTTP tunnel servlet 214 to the client 200:

[0137] Pull URL parameters=

[0138] "?ServerName=<ServerIdString>&Type=pull&ConnId=<connid>"

[0139] Each outgoing data packet (e.g. Packet Type=DATA_PACKET) may be assigned an incremental sequence number by the sender. The receiver may check the packet sequence number with its receive window. Any duplicate packets may be discarded.

[0140] After consuming a packet, the receiver may acknowledge the highest contiguous sequence number by sending the ACK packet as follows:

[0141] Packet Type=acknowledgement (e.g. ACK)

[0142] Connection ID=<connid>

[0143] Sequence=<sequence number of the data packet being acknowledged>

[0144] Receive Window Size=<remaining receive window capacity>

[0145] When an acknowledgement is received, the sender may update the round trip time for the connection. In one embodiment, a simple linear function of the computed round trip time may be used as the packet retransmission interval.

[0146] In one embodiment, when an acknowledgement packet reports a "Receive Window Size" of zero, the sender may stop sending further packets. The sender may send periodic repeat transmissions of the next packet to force the receiver to send a window update as soon as it is ready to receive more data. When the receiver indicates that it is ready to receive more data, the sender may resume sending packets.

[0147] In one embodiment, the HTTP Tunneling protocol may support the following runtime connection option. The client pull period is the duration (e.g. in seconds) of the idle period between consecutive pull requests sent by the client. If this value is positive, whenever the HTTP tunnel driver on the client side receives an empty response to its Pull request, the driver may sleep for the specified period before issuing another pull request. This may help conserve the servlet 214's resources and hence improve connection scalability.

[0148] Either client 200 or server 202 may set connection options. This may be used to control the runtime parameters associated with a connection. For example, this may be used to control how often packets are pulled from a Web server. As another example, this may be used to configure how long a client may remain inactive before the connection is dropped. The connection option values are part of the connection state information, and hence, in one embodiment, an HTTP packet including the following information may be used to update the connection options:

[0149] Packet Type=connection option packet (e.g. CONN_OPTION_PACKET)

Data = Option Type (integer)
 Option Value (integer)

[0150] In one embodiment, either end (client or server) may initiate connection shutdown by sending a connection close packet (e.g. Packet Type CONN_CLOSE_PACKET). Until both parties complete the connection shutdown, this packet may otherwise be treated like a normal data packet. This may help to ensure that all the data packets that are in

the pipeline ahead of the connection close packet are processed before the connection resources are destroyed. The remote end may consume all the data and acknowledge the connection close packet, at which point the connection is terminated and all the resources may be freed.

[0151] In one embodiment, a connection abort packet may be generated by the servlet when it realizes that either the server 202 or the client 200 has terminated ungracefully. The server termination may result in an IOException on the TCP connection between the servlet 214 and the server 202, and hence may be detected. In one embodiment, detecting client 200 termination may be handled as follows. If the servlet 214 does not receive a "pull" request from client 200 for a certain period of time, the servlet 214 may assume that the client 200 is not responding and thus may be down. The ungraceful connection shutdown may be achieved by sending a single packet with Packet Type connection abort packet (e.g. CONN_ABORT_PACKET) to the server 202 and possibly to the client 200 as well.

[0152] The following describes the initialization of the link between server 200 and HTTP tunneling servlet 214 according to one embodiment. The HTTP tunneling servlet 214 may listen on a fixed port number for TCP connections from servers. After the TCP connection is established to server 200, the server 200 may send the following information in an HTTP tunneling protocol packet to the servlet 214:

[0153] Packet Type: link initialization packet (e.g. LINK_INIT_PACKET)

ServerIdString (String)
ConnectionCount (Integer)
Data = Sequence of {ConnectionID (integer),
 ConnectionPullPeriod (integer)}

[0154] The ServerIdString may be used to establish the identity of the server 200. Clients may use this string to specify which server they want to talk to. This allows the use of a single servlet 214 as a gateway for multiple servers. The data portion of this packet may include information about any existing HTTP Tunnel connections. This allows HTTP tunnel connections to survive unexpected Web server/servlet engine failures or restarts.

[0155] Various embodiments may further include receiving, sending or storing instructions and/or data implemented in accordance with the foregoing description upon a carrier medium. Generally speaking, a carrier medium may include storage media or memory media such as magnetic or optical media, e.g., disk or CD-ROM, volatile or non-volatile media such as RAM (e.g. SDRAM, DDR SDRAM, RDRAM, SRAM, etc.), ROM, etc. as well as transmission media or signals such as electrical, electromagnetic, or digital signals, conveyed via a communication medium such as network and/or a wireless link.

[0156] In summary, a system and method for providing HTTP tunnel connections between entities such as clients and servers in a messaging system have been disclosed. It will be appreciated by those of ordinary skill having the benefit of this disclosure that the illustrative embodiments described above are capable of numerous variations without

departing from the scope and spirit of the invention. Various modifications and changes may be made as would be obvious to a person skilled in the art having the benefit of this disclosure. It is intended that the following claims be interpreted to embrace all such modifications and changes and, accordingly, the specifications and drawings are to be regarded in an illustrative rather than a restrictive sense.

What is claimed is:

1. A method comprising:

establishing a transport protocol tunnel connection from a first node in a messaging system to a second node in the messaging system;

generating a messaging system message on the first node;

generating one or more transport protocol packets, wherein the one or more transport protocol packets each includes at least a part of the messaging system message; and

transmitting the one or more transport protocol packets to the second node via the transport protocol tunnel connection;

wherein the transport protocol tunnel connection provides full-duplex transmission of messaging system messages between the first node and the second node, and wherein the transport protocol tunnel connection further provides delivery of the messaging system messages in the sequence in which the messaging system messages are generated.

2. The method as recited in claim 1, further comprising storing the messaging system message in a transmit buffer on the first node after said generating the messaging system message on the first node.

3. The method as recited in claim 1, wherein the transport protocol tunnel connection passes through a proxy server.

4. The method as recited in claim 3, wherein said transmitting the one or more transport protocol packets to the second node via the transport protocol tunnel connection comprises:

transmitting the one or more transport protocol packets from the first node to the proxy server; and

transmitting the one or more transport protocol packets from the proxy server to the second node.

5. The method as recited in claim 1, wherein the transport protocol tunnel connection passes through at least one firewall.

6. The method as recited in claim 1, wherein the transport protocol tunnel connection is established through a network.

7. The method as recited in claim 6, wherein the network is the Internet.

8. The method as recited in claim 1, wherein the first node is a client in the messaging system, and wherein the second node is a broker in the messaging system.

9. The method as recited in claim 1, wherein the transport protocol tunnel connection passes through a third node, and wherein, in said transmitting the one or more transport protocol packets to the second node, the method further comprises:

transmitting the one or more transport protocol packets to the third node; and

the third node forwarding the one or more transport protocol packets to the second node.

10. The method as recited in claim 9, wherein the one or more transport protocol packets are forwarded to the second node via a Transmission Control Protocol (TCP) connection portion of the transport protocol tunnel connection between the third node and the second node.

11. The method as recited in claim 9, wherein the third node is a Web server.

12. The method as recited in claim 1, wherein the transport protocol tunnel connection passes through a proxy server and a Web server, and wherein said transmitting the one or more transport protocol packets to the second node via the transport protocol tunnel connection comprises:

transmitting the one or more transport protocol packets from the first node to the proxy server;

transmitting the one or more transport protocol packets from the proxy server to the Web server; and

the Web server forwarding the one or more transport protocol packets to the second node.

13. The method as recited in claim 12, wherein the transport protocol tunnel connection passes through at least one firewall between the proxy server and the Web server.

14. The method as recited in claim 1, wherein the one or more transport protocol packets include messaging system message sequence information configured for use in processing two or more messaging system messages in sequence.

15. The method as recited in claim 1, further comprising:

receiving the transmitted one or more transport protocol packets on the second node; and

storing the messaging system message from the one or more transport protocol packets in a receive buffer on the second node.

16. The method as recited in claim 1, further comprising:

receiving the transmitted one or more transport protocol packets on the second node;

the second node generating an acknowledgement transport protocol packet to indicate successful receipt of the one or more transport protocol packets including the messaging system message; and

transmitting the acknowledgement transport protocol packet to the first node via the transport protocol tunnel connection.

17. The method as recited in claim 16, further comprising:

storing the messaging system message from the received one or more transport protocol packets in a receive buffer on the second node;

wherein the acknowledgement transport protocol packet includes information indicating available space in the receive buffer, and wherein the information indicating available space in the receive buffer is configured for use in flow control of messaging system messages transmitted from the first node to the second node.

18. The method as recited in claim 17, further comprising:

receiving the transmitted acknowledgement transport protocol packet on the first node;

- generating one or more messaging system messages on the first node;
- storing the one or more messaging system messages in a transmit buffer on the first node;
- determining from the information indicating available space in the receive buffer included in the received acknowledgement transport protocol packet if there is space available to receive the one or more messaging system messages on the second node;
- if said determining indicates there is space available to store the one or more messaging system messages in the receive buffer of the second node:
- generating a second one or more transport protocol packets, wherein the second one or more transport protocol packets include the one or more messaging system messages; and
 - transmitting the second one or more transport protocol packets to the second node via the transport protocol tunnel connection; and
- if said determining indicates there is not space available to store the second messaging system message in the receive buffer of the second node, inhibiting generating the second one or more transport protocol packets including the one or more messaging system messages.
19. The method as recited in claim 18, further comprising:
- the first node receiving a transport protocol packet indicating available space in the receive buffer of the second node;
 - determining from the information indicating available space in the receive buffer included in the received transport protocol packet that there is space available to receive the one or more messaging system messages on the second node;
 - generating the second one or more transport protocol packets, wherein the second one or more transport protocol packets include the one or more messaging system messages; and
 - transmitting the second one or more transport protocol packets to the second node via the transport protocol tunnel connection.
20. The method as recited in claim 16, wherein the transport protocol tunnel connection passes through a third node, and wherein, in said transmitting the acknowledgement transport protocol packet to the first node, the method further comprises:
- transmitting the acknowledgement transport protocol packet to the third node; and
 - storing the acknowledgement transport protocol packet in a transport protocol packet buffer on the third node.
21. The method as recited in claim 20, wherein, in said transmitting the acknowledgement transport protocol packet to the first node, the method further comprises:
- the first node transmitting a transport protocol request packet to the third node; and
 - the third node transmitting the acknowledgement transport protocol packet stored in the transport protocol packet buffer to the first node via the transport protocol tunnel connection in response to the transport protocol request packet.
22. The method as recited in claim 21, wherein the acknowledgement transport protocol packet is transmitted to the third node via a Transmission Control Protocol (TCP) connection portion of the transport protocol tunnel connection.
23. The method as recited in claim 16, wherein the transport protocol tunnel connection passes through a third node, and wherein, in said transmitting the acknowledgement transport protocol packet to the first node, the method further comprises:
- transmitting the acknowledgement transport protocol packet to the third node; and
 - the third node forwarding the acknowledgement transport protocol packet to the first node.
24. The method as recited in claim 23, wherein the acknowledgement transport protocol packet are forwarded to the first node via a Transmission Control Protocol (TCP) connection portion of the transport protocol tunnel connection.
25. The method as recited in claim 1, wherein the first node is a server in the messaging system, and wherein the second node is a client in the messaging system.
26. The method as recited in claim 1, wherein the transport protocol tunnel connection passes through a third node, and wherein, in said transmitting the one or more transport protocol packets to the second node via the transport protocol tunnel connection, the method further comprises:
- transmitting the one or more transport protocol packets to the third node; and
 - storing the one or more transport protocol packets in a transport protocol packet buffer on the third node.
27. The method as recited in claim 26, wherein, in said transmitting the one or more transport protocol packets to the second node via the transport protocol tunnel connection, the method further comprises:
- the second node sending one or more transport protocol request packets to the third node; and
 - the third node transmitting the one or more transport protocol packets stored in the transport protocol packet buffer to the second node via the transport protocol tunnel connection in response to the one or more transport protocol request packets.
28. The method as recited in claim 26, wherein the third node is a Web server.
29. The method as recited in claim 1, wherein the transport protocol is Hypertext Transport Protocol (HTTP).
30. The method as recited in claim 1, wherein the transport protocol is one of UDP (User Datagram Protocol), IrDA (Infrared Data Association), SNA (Systems Network Architecture), IPX (Internetwork Packet eXchange), and Bluetooth.
31. A method comprising:
- establishing a Hypertext Transport Protocol (HTTP) tunnel connection from a first node in a messaging system to a second node in the messaging system;
 - generating a messaging system message on the first node;

generating one or more HTTP packets, wherein the one or more HTTP packets each includes at least a part of the messaging system message; and

transmitting the one or more HTTP packets to the second node via the HTTP tunnel connection;

wherein the HTTP tunnel connection provides full-duplex transmission of messaging system messages between the first node and the second node, and wherein the HTTP tunnel connection further provides delivery of the messaging system messages in the sequence in which the messaging system messages are generated.

32. The method as recited in claim 31, wherein the first node is a client in the messaging system, wherein the HTTP tunnel connection passes through a proxy server, and wherein said transmitting the one or more HTTP packets to the second node via the HTTP tunnel connection comprises:

transmitting the one or more HTTP packets from the client to the proxy server; and

transmitting the one or more HTTP packets from the proxy server to the second node.

33. The method as recited in claim 31, wherein the HTTP tunnel connection is established through the Internet, and wherein the HTTP tunnel connection passes through at least one firewall.

34. The method as recited in claim 31, wherein the first node is a client in the messaging system, and wherein the second node is a broker in the messaging system.

35. The method as recited in claim 31, wherein the HTTP tunnel connection passes through a Web server, wherein the second node is a broker in the messaging system, and wherein, in said transmitting the one or more HTTP packets to the second node, the method further comprises:

transmitting the one or more HTTP packets to the Web server; and

the Web server forwarding the one or more HTTP packets to the broker via a Transmission Control Protocol (TCP) connection portion of the HTTP tunnel connection between the Web server and the broker.

36. The method as recited in claim 31, wherein the first node is a client in the messaging system, wherein the second node is a broker in the messaging system, wherein the HTTP tunnel connection passes through a proxy server and a Web server, and wherein said transmitting the one or more HTTP packets to the broker via the HTTP tunnel connection comprises:

transmitting the one or more HTTP packets from the client to the proxy server;

transmitting the one or more HTTP packets from the proxy server to the Web server; and

the Web server forwarding the one or more HTTP packets to the broker;

wherein the HTTP tunnel connection passes through at least one firewall between the proxy server and the Web server.

37. The method as recited in claim 31, wherein the one or more HTTP packets include messaging system message sequence information configured for use in processing two or more messaging system messages in sequence.

38. The method as recited in claim 31, further comprising:

receiving the transmitted one or more HTTP packets on the second node;

storing the messaging system message from the one or more HTTP packets in a receive buffer on the second node;

the second node generating an acknowledgement HTTP packet to indicate successful receipt of the one or more HTTP packets including the messaging system message; and

transmitting the acknowledgement HTTP packet to the first node via the HTTP tunnel connection.

39. The method as recited in claim 38, wherein the acknowledgement HTTP packet includes information indicating available space in the receive buffer, the method further comprising:

receiving the transmitted acknowledgement HTTP packet on the first node;

generating one or more messaging system messages on the first node;

storing the one or more messaging system messages in a transmit buffer on the first node;

determining from the information indicating available space in the receive buffer included in the received acknowledgement HTTP packet that there is not space available to receive the one or more messaging system messages on the second node;

the first node receiving an HTTP packet from the second node indicating available space in the receive buffer of the second node;

determining from the information indicating available space in the receive buffer included in the received HTTP packet that there is space available to receive the one or more messaging system messages on the second node;

generating a second one or more HTTP packets, wherein the second one or more HTTP packets include the one or more messaging system messages; and

transmitting the second one or more HTTP packets to the second node via the HTTP tunnel connection.

40. The method as recited in claim 38, wherein the first node is a client in the messaging system, wherein the HTTP tunnel connection passes through a Web server, and wherein, in said transmitting the acknowledgement HTTP packet to the first node, the method further comprises:

transmitting the acknowledgement HTTP packet to the Web server;

storing the acknowledgement HTTP packet in an HTTP packet buffer on the Web server;

the client sending an HTTP request packet to the Web server; and

the Web server transmitting the acknowledgement HTTP packet stored in the HTTP packet buffer to the client via the HTTP tunnel connection in response to the HTTP request packet.

41. The method as recited in claim 38, wherein the first node is a broker in the messaging system, wherein the HTTP tunnel connection passes through a Web server, and wherein, in said transmitting the acknowledgement HTTP packet to the first node, the method further comprises:

transmitting the acknowledgement HTTP packet to the Web server; and

the Web server forwarding the acknowledgement HTTP packet to the first node via a Transmission Control Protocol (TCP) connection portion of the HTTP tunnel connection.

42. The method as recited in claim 31, wherein the first node is a server in the messaging system, and wherein the second node is a client in the messaging system.

43. The method as recited in claim 31, wherein the second node is a client in the messaging system, wherein the HTTP tunnel connection passes through a Web server, and wherein, in said transmitting the one or more HTTP packets to the second node via the HTTP tunnel connection, the method further comprises:

transmitting the one or more HTTP packets to the Web server;

storing the one or more HTTP packets in an HTTP packet buffer on the Web server;

the client sending one or more HTTP request packets to the Web server; and

the Web server transmitting the one or more HTTP packets stored in the HTTP packet buffer to the client via the HTTP tunnel connection in response to the one or more HTTP request packets.

44. A method comprising:

establishing a transport protocol tunnel connection from a first node in a messaging system to a second node in the messaging system;

generating a sequence of messaging system messages on the first node;

generating a plurality of transport protocol packets on the first node, wherein each of the transport protocol packets includes at least a part of one of the sequence of messaging system messages, and wherein each of the transport protocol packets includes sequence information for the particular messaging system message;

transmitting the plurality of transport protocol packets to the second node in the messaging system via the transport protocol tunnel connection;

receiving the plurality of transport protocol packets on the second node; and

processing the sequence of messaging system messages on the second node, wherein said processing uses the sequence information for the plurality of messaging system messages in the plurality of transport protocol packets.

45. The method as recited in claim 44, wherein the transport protocol tunnel connection is established through the Internet, and wherein the transport protocol tunnel connection passes through at least one firewall.

46. The method as recited in claim 44, wherein the first node is a client in the messaging system, wherein the second node is a broker in the messaging system.

47. The method as recited in claim 44, wherein the second node is a broker in the messaging system, wherein the transport protocol tunnel connection passes through a Web server, and wherein said transmitting the plurality of transport protocol packets to the second node in the messaging system via the transport protocol tunnel connection comprises:

transmitting the plurality of transport protocol packets from the first node to the Web server; and

the Web server forwarding the plurality of transport protocol packets to the broker;

wherein the transport protocol tunnel connection passes through at least one firewall between the proxy server and the Web server.

48. The method as recited in claim 44, wherein the first node is a broker in the messaging system, wherein the second node is a client in the messaging system.

49. The method as recited in claim 44, wherein the second node is a client in the messaging system, wherein the transport protocol tunnel connection passes through a Web server, and wherein, in said transmitting the plurality of transport protocol packets to the second node in the messaging system via the transport protocol tunnel connection comprises:

transmitting the plurality of transport protocol packets to the Web server;

storing the plurality of transport protocol packets in a transport protocol packet buffer on the Web server;

the client sending one or more transport protocol request packets to the Web server; and

the Web server transmitting the plurality of transport protocol packets stored in the transport protocol packet buffer to the client via the transport protocol tunnel connection in response to the one or more transport protocol request packets.

50. The method as recited in claim 44, further comprising:

storing the sequence of messaging system messages from the received transport protocol packets in a receive buffer on the second node;

the second node generating an acknowledgement transport protocol packet for each of the received transport protocol packets to indicate successful receipt of the transport protocol packets including the sequence of messaging system messages; and

transmitting the acknowledgement transport protocol packets to the first node via the transport protocol tunnel connection;

wherein each of the acknowledgement transport protocol packets includes information indicating available space in the receive buffer, wherein the information indicating available space in the receive buffer is configured for use in flow control of messaging system messages transmitted from the first node to the second node.

51. The method as recited in claim 44, wherein the transport protocol is Hypertext Transport Protocol (HTTP).

52. A method comprising:

establishing a transport protocol tunnel connection from a first node in a messaging system to a second node in the messaging system;

the first node receiving a first transport protocol packet from the second node indicating available space in a receive buffer of the second node;

generating one or more messaging system messages on the first node;

storing the one or more messaging system messages in a transmit buffer on the first node;

determining from the information indicating available space in the receive buffer included in the received acknowledgement HTTP packet that there is not space available to receive the one or more messaging system messages on the second node;

the first node receiving a second transport protocol packet from the second node indicating available space in the receive buffer of the second node;

determining from the information indicating available space in the receive buffer included in the received second transport protocol packet that there is space available to receive the one or more messaging system messages on the second node;

generating one or more transport protocol packets, wherein the second one or more transport protocol packets include the generated one or more messaging system messages; and

transmitting the one or more transport protocol packets to the second node via the transport protocol tunnel connection.

53. The method as recited in claim 52, wherein the transport protocol tunnel connection is established through the Internet, and wherein the transport protocol tunnel connection passes through at least one firewall.

54. The method as recited in claim 52, wherein the first node is a client in the messaging system, wherein the second node is a broker in the messaging system.

55. The method as recited in claim 52, wherein the first node is a broker in the messaging system, wherein the second node is a client in the messaging system.

56. The method as recited in claim 52, further comprising:

receiving the one or more transport protocol packets on the second node;

storing the one or more messaging system messages from the received one or more transport protocol packets in the receive buffer on the second node;

the second node generating one or more acknowledgement transport protocol packets to indicate successful receipt of the one or more transport protocol packets including the one or more of messaging system messages; and

transmitting the one or more acknowledgement transport protocol packets to the first node via the transport protocol tunnel connection;

wherein each of the acknowledgement transport protocol packets includes information indicating available space in the receive buffer.

57. The method as recited in claim 52, wherein the one or more transport protocol packets include messaging system message sequence information configured for use in processing two or more messaging system messages in sequence.

58. The method as recited in claim 52, wherein the transport protocol is Hypertext Transport Protocol (HTTP).

59. A messaging system comprising:

a first node comprising a first memory;

a second node comprising a second memory;

wherein the first memory comprises first program instructions executable within the first node to:

establish a transport protocol tunnel connection from the first node to the second node through a network;

generate a messaging system message;

generate one or more transport protocol packets, wherein the one or more transport protocol packets each includes at least a part of the messaging system message; and

transmit the one or more transport protocol packets to the second node via the transport protocol tunnel connection;

wherein the transport protocol tunnel connection provides full-duplex transmission of messaging system messages between the first node and the second node, and wherein the transport protocol tunnel connection further provides delivery of the messaging system messages in the sequence in which the messaging system messages are generated.

60. The messaging system as recited in claim 59, wherein the first node further comprises a transmit buffer, wherein the first program instructions are further executable within the first node to store the messaging system message in the transmit buffer on the first node after said generating the messaging system message.

61. The messaging system as recited in claim 59, wherein the transport protocol tunnel connection passes through a proxy server, and wherein, in said transmitting the one or more transport protocol packets to the second node via the transport protocol tunnel connection, the first program instructions are further executable within the first node to transmit the one or more transport protocol packets from the first node to the proxy server, wherein the proxy server is configured to transmit the one or more transport protocol packets to the second node.

62. The messaging system as recited in claim 59, wherein the transport protocol tunnel connection passes through at least one firewall.

63. The messaging system as recited in claim 59, wherein the transport protocol tunnel connection is established through the Internet.

64. The messaging system as recited in claim 59, wherein the first node is a client in the messaging system, and wherein the second node is a broker in the messaging system.

65. The messaging system as recited in claim 59, wherein the messaging system further comprises:

a third node comprising a third memory;

wherein the transport protocol tunnel connection passes through the third node, and wherein, in said transmitting the one or more transport protocol packets to the second node, the first program instructions are further executable within the first node to:

transmit the one or more transport protocol packets to the third node;

wherein the third memory comprises third program instructions executable within the third node to:

receive the one or more transport protocol packets from the first node; and

forward the one or more received transport protocol packets to the second node.

66. The messaging system as recited in claim 65, wherein the one or more transport protocol packets are forwarded from the third node to the second node via a Transmission Control Protocol (TCP) connection portion of the transport protocol tunnel connection between the third node and the second node.

67. The messaging system as recited in claim 66, wherein the transport protocol tunnel connection passes through at least one firewall between the first node and the third node.

68. The messaging system as recited in claim 59, wherein the one or more transport protocol packets include messaging system message sequence information configured for use in processing two or more messaging system messages in sequence.

69. The messaging system as recited in claim 59, wherein the second memory comprises second program instructions executable within the second node to:

receive the transmitted one or more transport protocol packets;

generate an acknowledgement transport protocol packet to indicate successful receipt of the one or more transport protocol packets including the messaging system message; and

transmit the acknowledgement transport protocol packet to the first node via the transport protocol tunnel connection.

70. The messaging system as recited in claim 69, wherein the second node further comprises a receive buffer, wherein the second program instructions are further executable within the second node to:

store the messaging system message from the received one or more transport protocol packets in the receive buffer of the second node;

wherein the acknowledgement transport protocol packet includes information indicating available space in the receive buffer of the second node.

71. The messaging system as recited in claim 70, wherein the first node further comprises a transmit buffer, wherein the first program instructions are further executable within the first node to:

receive the transmitted acknowledgement transport protocol packet;

generate one or more messaging system messages;

store the one or more messaging system messages in the transmit buffer on the first node;

from the information indicating available space in the receive buffer included in the received acknowledgement transport protocol packet, determine if there is space available to receive the one or more messaging system messages on the second node;

if said determining indicates there is space available to store the one or more messaging system messages in the receive buffer of the second node:

generate a second one or more transport protocol packets, wherein the second one or more transport protocol packets include the one or more messaging system messages; and

transmit the second one or more transport protocol packets to the second node via the transport protocol tunnel connection; and

if said determining indicates there is not space available to store the second messaging system message in the receive buffer of the second node, inhibit generating the second one or more transport protocol packets including the one or more messaging system messages.

72. The messaging system as recited in claim 71, wherein the first program instructions are further executable within the first node to:

receive a transport protocol packet indicating available space in the receive buffer of the second node;

from the information indicating available space in the receive buffer included in the received transport protocol packet, determine that there is space available to receive the one or more messaging system messages on the second node;

generate the second one or more transport protocol packets, wherein the second one or more transport protocol packets include the one or more messaging system messages; and

transmit the second one or more transport protocol packets to the second node via the transport protocol tunnel connection.

73. The messaging system as recited in claim 69, further comprising:

a third node comprising:

a third memory; and

a transport protocol packet buffer;

wherein the transport protocol tunnel connection passes through the third node, wherein the third memory comprises third program instructions executable within the third node to:

receive the acknowledgement transport protocol packet transmitted to the first node via the transport protocol tunnel connection from the second node; and

store the received acknowledgement transport protocol packet in the transport protocol packet buffer.

74. The messaging system as recited in claim 73, wherein the first program instructions are further executable within the first node to:

transmit a transport protocol request packet to the third node; and

wherein the third program instructions are further executable within the third node to:

receive the transport protocol request packet from the first node; and

transmit the acknowledgement transport protocol packet stored in the transport protocol packet buffer to the first node via the transport protocol tunnel connection in response to the received transport protocol request packet.

75. The messaging system as recited in claim 69, further comprising:

a third node comprising a third memory, wherein the transport protocol tunnel connection passes through the third node, wherein the third memory comprises third program instructions executable within the third node to:

receive the acknowledgement transport protocol packet transmitted to the first node via the transport protocol tunnel connection from the second node; and

forward the acknowledgement transport protocol packet to the first node.

76. The messaging system as recited in claim 59, wherein the first node is a server in the messaging system, and wherein the second node is a client in the messaging system.

77. The messaging system as recited in claim 59, further comprising:

a third node comprising:

a third memory; and

a transport protocol packet buffer;

wherein the transport protocol tunnel connection passes through the third node, wherein the third memory comprises third program instructions executable within the third node to:

receive the one or more transport protocol packets transmitted to the second node via the transport protocol tunnel connection from the first node; and

store the one or more transport protocol packets in the transport protocol packet buffer on the third node.

wherein the second program instructions are further executable within the second node to transmit one or more transport protocol request packets to the third node; and

wherein the third program instructions are further executable within the third node to:

receive the one or more transmitted transport protocol request packets; and

transmit the one or more transport protocol packets stored in the transport protocol packet buffer to the second node via the transport protocol tunnel connection in response to the received one or more transport protocol request packets.

78. The messaging system as recited in claim 59, wherein the transport protocol is Hypertext Transport Protocol (HTTP).

79. The messaging system as recited in claim 59, wherein the transport protocol is one of UDP (User Datagram Protocol), IrDA (Infrared Data Association), SNA (Systems Network Architecture), IPX (Internetwork Packet eXchange), and Bluetooth.

80. A carrier medium comprising program instructions, wherein the program instructions are computer-executable to implement:

establishing a transport protocol tunnel connection from a first node in a messaging system to a second node in the messaging system;

generating a messaging system message on the first node;

generating one or more transport protocol packets, wherein the one or more transport protocol packets each includes at least a part of the messaging system message; and

transmitting the one or more transport protocol packets to the second node via the transport protocol tunnel connection;

wherein the transport protocol tunnel connection provides full-duplex transmission of messaging system messages between the first node and the second node, and wherein the transport protocol tunnel connection further provides delivery of the messaging system messages in the sequence in which the messaging system messages are generated.

81. The carrier medium as recited in claim 80, wherein the transport protocol tunnel connection passes through a Web server, wherein the second node is a broker in the messaging system, and wherein, in said transmitting the one or more transport protocol packets to the second node, the program instructions are further computer-executable to implement:

transmitting the one or more transport protocol packets to the Web server; and

the Web server forwarding the one or more transport protocol packets to the broker via a Transmission Control Protocol (TCP) connection portion of the transport protocol tunnel connection between the Web server and the broker.

82. The carrier medium as recited in claim 80, wherein the first node is a client in the messaging system, wherein the second node is a broker in the messaging system, wherein the transport protocol tunnel connection passes through a proxy server and a Web server, and wherein, in said transmitting the one or more transport protocol packets to the broker via the transport protocol tunnel connection, the program instructions are further computer-executable to implement:

transmitting the one or more transport protocol packets from the client to the proxy server;

transmitting the one or more transport protocol packets from the proxy server to the Web server; and

the Web server forwarding the one or more transport protocol packets to the broker;

wherein the transport protocol tunnel connection passes through at least one firewall between the proxy server and the Web server.

83. The carrier medium as recited in claim 80, wherein the acknowledgement transport protocol packet includes information indicating available space in the receive buffer, and wherein the program instructions are further computer-executable to implement:

receiving the transmitted one or more transport protocol packets on the second node;

storing the messaging system message from the one or more transport protocol packets in a receive buffer on the second node;

generating on the second node an acknowledgement transport protocol packet to indicate successful receipt of the one or more transport protocol packets including the messaging system message;

transmitting the acknowledgement transport protocol packet to the first node via the transport protocol tunnel connection;

receiving the transmitted acknowledgement transport protocol packet on the first node;

generating one or more messaging system messages on the first node;

storing the one or more messaging system messages in a transmit buffer on the first node;

determining from the information indicating available space in the receive buffer included in the received acknowledgement transport protocol packet that there is space available to receive the one or more messaging system messages on the second node;

generating a second one or more transport protocol packets, wherein the second one or more transport protocol packets include the one or more messaging system messages; and

transmitting the second one or more transport protocol packets to the second node via the transport protocol tunnel connection.

84. The carrier medium as recited in claim 80, wherein the first node is a client in the messaging system, wherein the transport protocol tunnel connection passes through a Web server, and wherein the program instructions are further computer-executable to implement:

receiving the transmitted one or more transport protocol packets on the second node;

storing the messaging system message from the one or more transport protocol packets in a receive buffer on the second node;

the second node generating an acknowledgement transport protocol packet to indicate successful receipt of the one or more transport protocol packets including the messaging system message;

transmitting the acknowledgement transport protocol packet to the Web server;

storing the acknowledgement transport protocol packet in a transport protocol packet buffer on the Web server;

the client sending a transport protocol request packet to the Web server; and

the Web server transmitting the acknowledgement transport protocol packet stored in the transport protocol packet buffer to the client via the transport protocol tunnel connection in response to the transport protocol request packet.

85. The carrier medium as recited in claim 80, wherein the first node is a broker in the messaging system, wherein the transport protocol tunnel connection passes through a Web server, and wherein the program instructions are further computer-executable to implement:

receiving the transmitted one or more transport protocol packets on the second node;

storing the messaging system message from the one or more transport protocol packets in a receive buffer on the second node;

the second node generating an acknowledgement transport protocol packet to indicate successful receipt of the one or more transport protocol packets including the messaging system message;

transmitting the acknowledgement transport protocol packet to the Web server; and

the Web server forwarding the acknowledgement transport protocol packet to the first node via a Transmission Control Protocol (TCP) connection portion of the transport protocol tunnel connection.

86. The carrier medium as recited in claim 80, wherein the second node is a client in the messaging system, wherein the transport protocol tunnel connection passes through a Web server, and wherein, in said transmitting the one or more transport protocol packets to the second node via the transport protocol tunnel connection, the program instructions are further computer-executable to implement:

transmitting the one or more transport protocol packets to the Web server;

storing the one or more transport protocol packets in a transport protocol packet buffer on the Web server;

the client sending one or more transport protocol request packets to the Web server; and

the Web server transmitting the one or more transport protocol packets stored in the transport protocol packet buffer to the client via the transport protocol tunnel connection in response to the one or more transport protocol request packets.

87. The carrier medium as recited in claim 80, wherein the transport protocol is Hypertext Transport Protocol (HTTP).

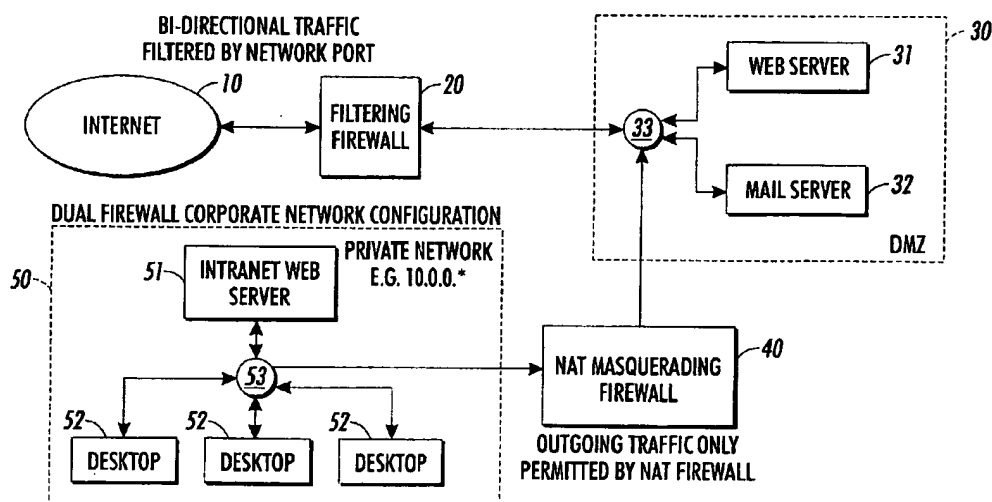
* * * * *



US 20020161904A1

(19) **United States**(12) **Patent Application Publication** (10) Pub. No.: **US 2002/0161904 A1**
(43) Pub. Date: **Oct. 31, 2002**(54) **EXTERNAL ACCESS TO PROTECTED
DEVICE ON PRIVATE NETWORK**(22) Filed: **Apr. 30, 2001****Publication Classification**(75) Inventors: **Gavan Tredoux, Rochester, NY (US);
Xin Xu, Palo Alto, CA (US); Bruce C.
Lyon, Victor, NY (US); Randy L.
Cain, Plano, TX (US)**(51) Int. Cl.⁷ **G06F 15/16; G06F 12/14**(52) U.S. Cl. **709/229; 713/200**Correspondence Address:
**Patent Documentation Center
Xerox Corporation
Xerox Square 20th Floor
100 Clinton Ave. S.
Rochester, NY 14644 (US)**(57) **ABSTRACT**

A scheme allowing communication between a network device on a protected network and an external network device outside the protected network using "reverse proxying." A proxy server receives incoming data on behalf of the protected network device, which data is retrieved by a proxy agent that periodically polls the proxy server to see if any data awaits retrieval.

(73) Assignee: **Xerox Corporation**(21) Appl. No.: **09/845,104**

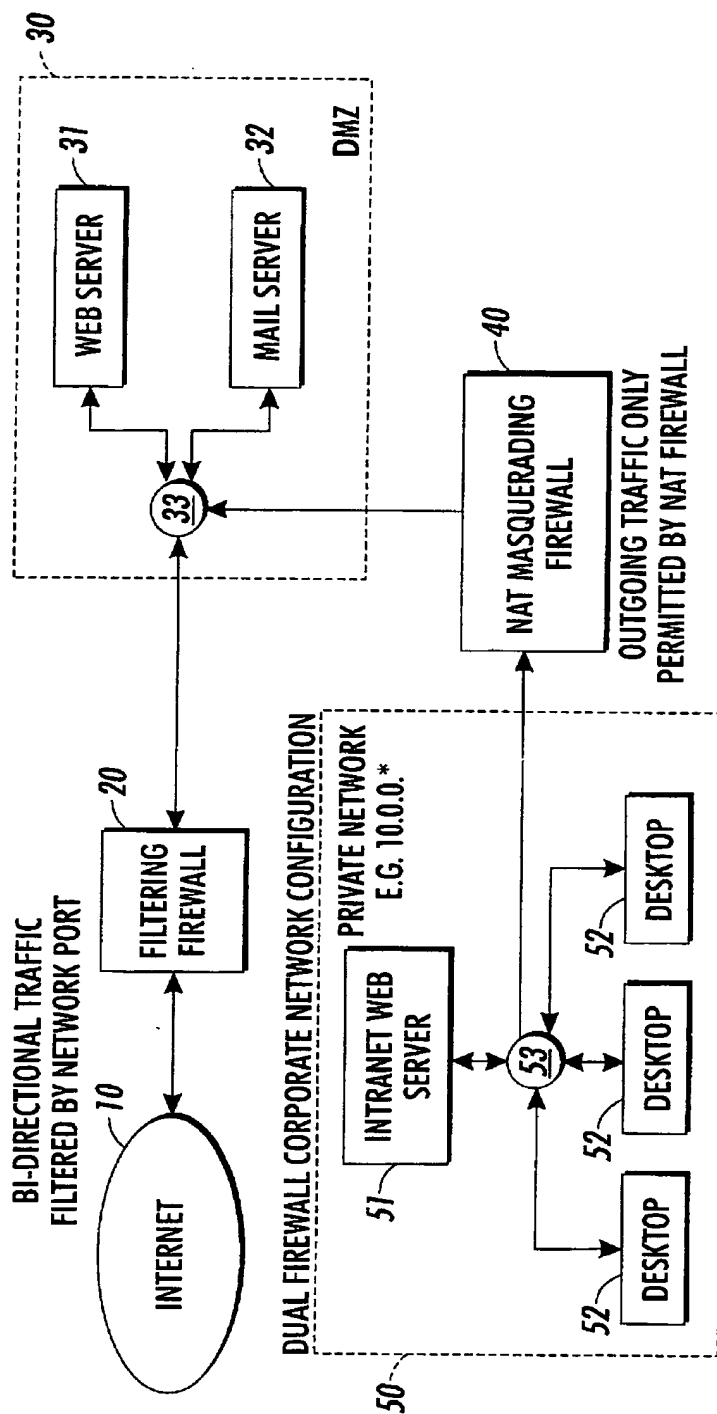


FIG. 1

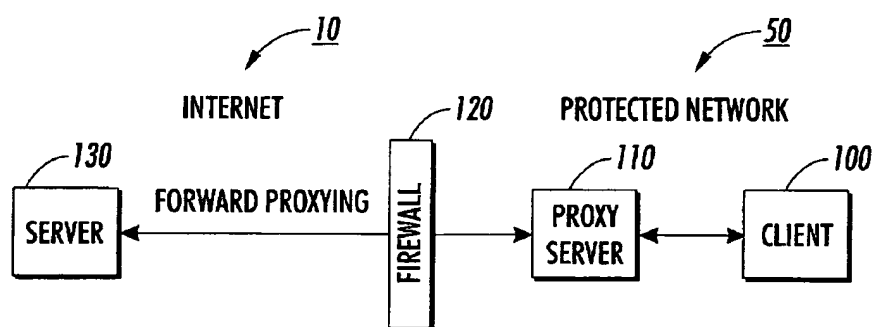


FIG. 2A

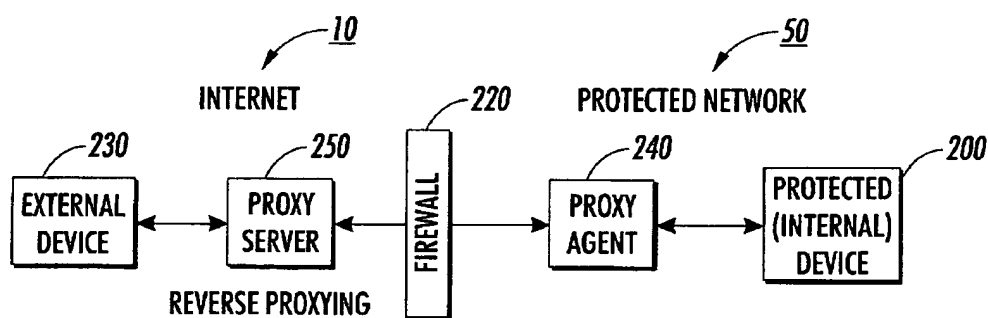


FIG. 2B

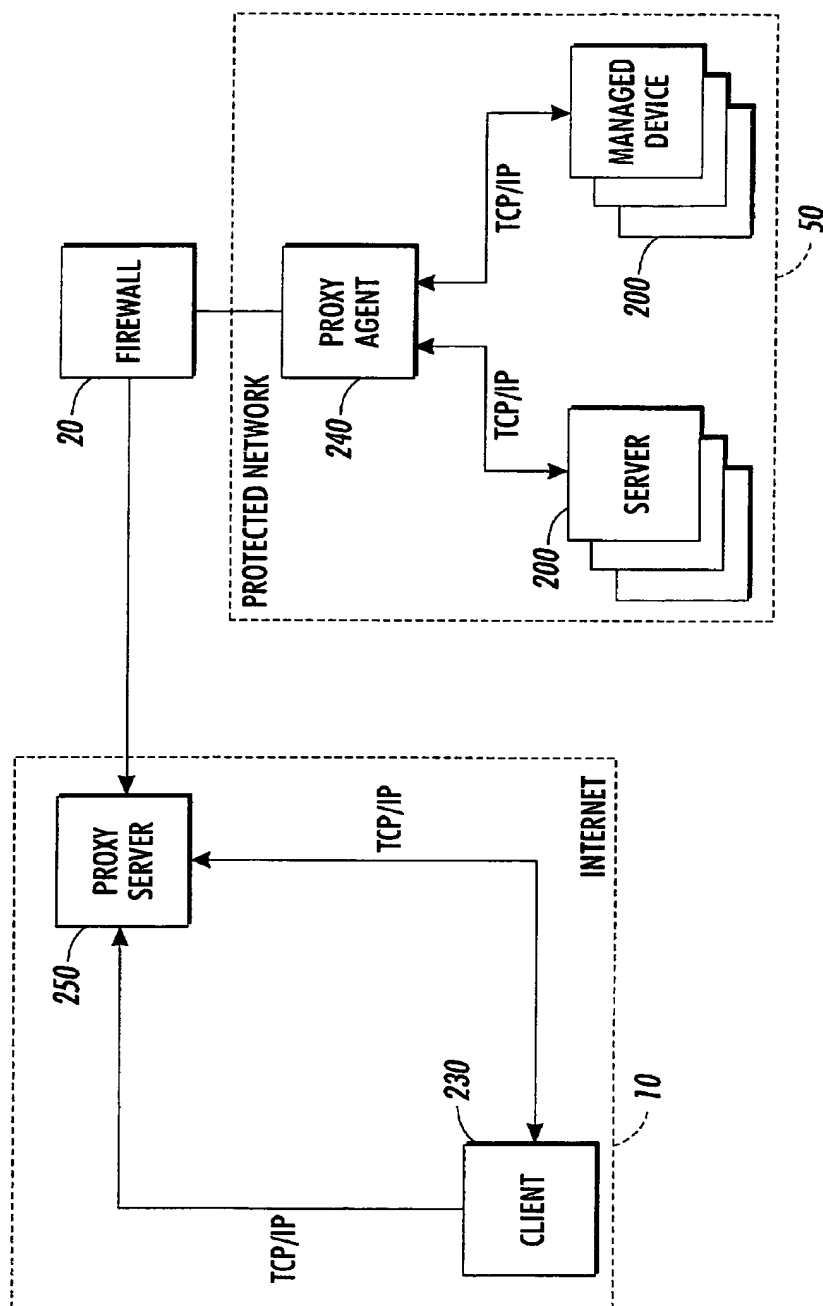


FIG. 3

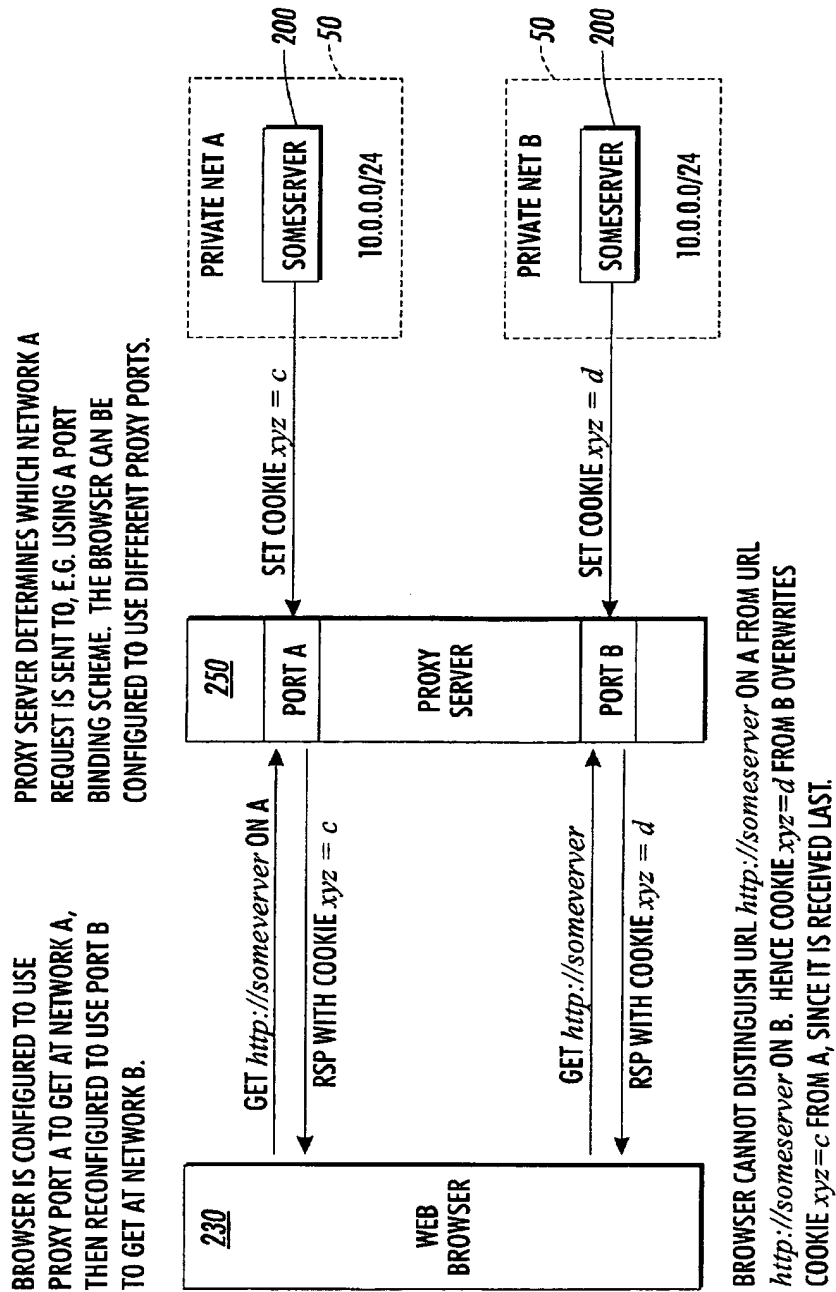


FIG. 4

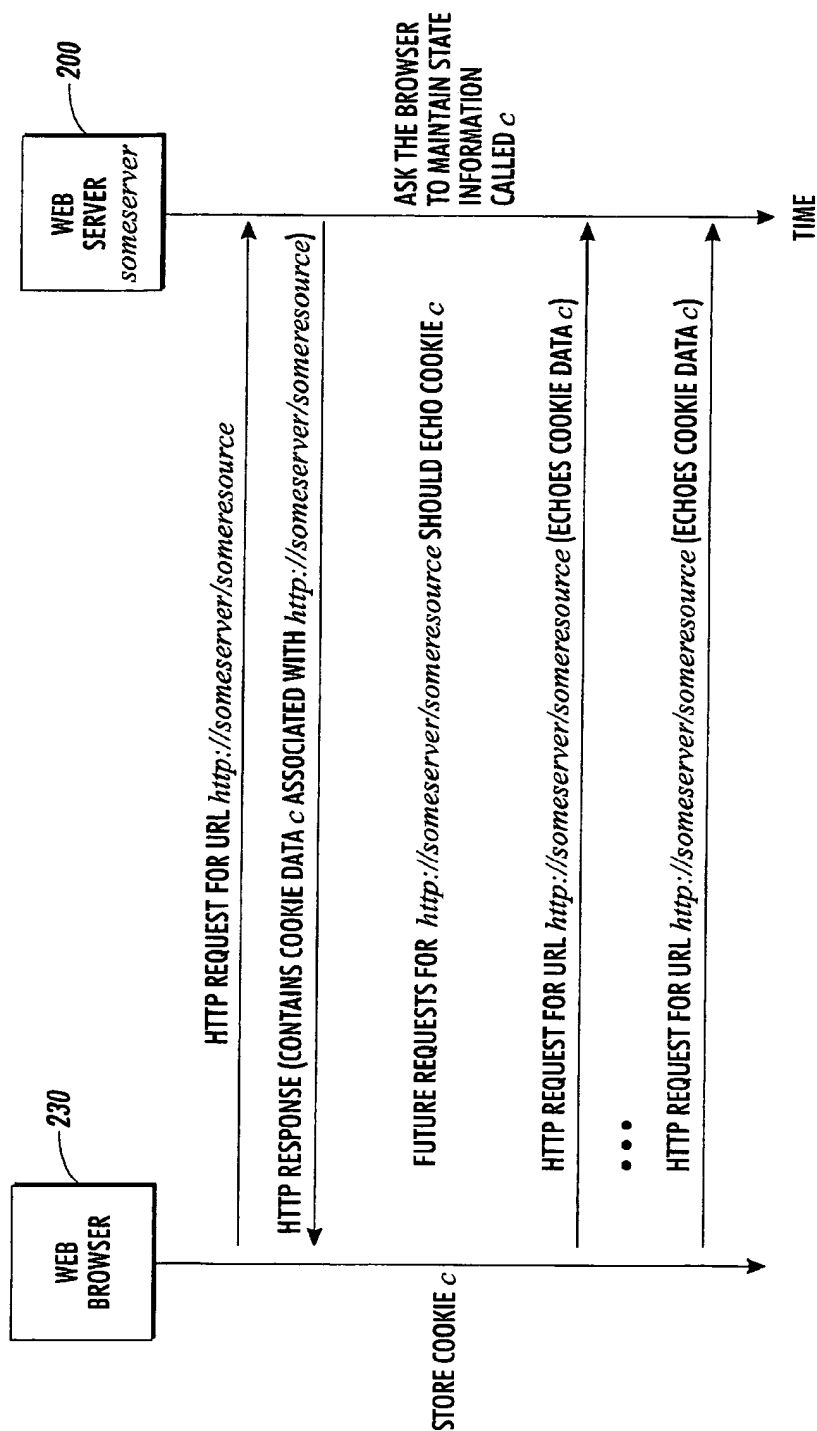


FIG. 5

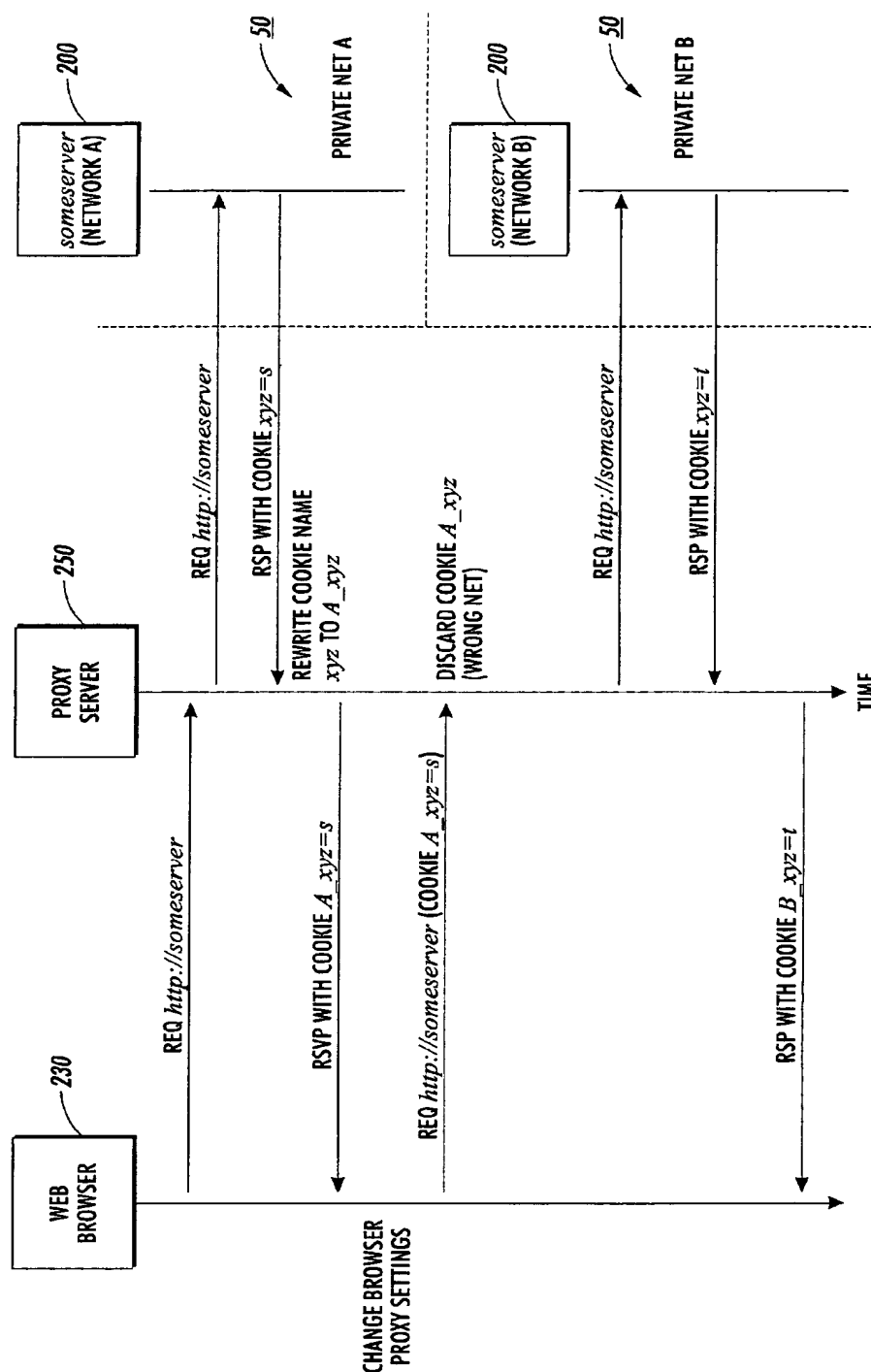


FIG. 6

EXTERNAL ACCESS TO PROTECTED DEVICE ON PRIVATE NETWORK

FIELD OF THE INVENTION

[0001] The present invention relates to protection and access protocols for networks such as computer networks and the like. In particular, the present invention relates to schemes allowing access to and from devices on protected networks from outside the protected networks.

BACKGROUND OF THE INVENTION

[0002] Networks connected to the Internet rely on firewalls and proxy servers to protect the networks against intrusion by unauthorized persons. Firewalls typically allow only incoming connections to designated machines and/or via particular protocols (TCP/IP, HTTP, FTP, etc.), disallowing all other traffic. Firewalls can also restrict traffic from the network to the Internet, as can outgoing proxy servers, by restricting destinations and/or protocols. However, these security restrictions often frustrate some uses of the Internet for legitimate purposes. For example, remote network equipment diagnosis and service is severely impaired, if not completely disabled, by firewalls.

[0003] Some firewalls can be modified and/or reconfigured to permit the traffic entry, but this can require the purchase of additional hardware and/or software. The cost associated with hardware and/or software purchase, combined with the difficulty of effecting a change in corporate policies regarding network security, would likely be a significant obstacle to the realization of such modifications. In addition, many firewalls and/or routers employ address masquerading and network address translation (NAT). Masquerading and NAT allow the use of internal network address spaces, but typically prevent incoming traffic from reaching the internal addresses since the internal addresses are non-routable and non-unique. No commercially-used or -available technique appears to solve all of these problems without modification of firewall/proxy server configurations, firewall/proxy server capabilities, and/or network security policies. For example, many virtual private network (VPN) schemes provide secure access between private networks via the Internet, but all require extensive modifications to the firewalls, proxy servers, and/or security policies of the connected networks.

SUMMARY OF THE INVENTION

[0004] Various embodiments of the invention allow traffic from outside a protected network to connect to an internal network device of the protected network through a firewall configured to protect the network. For example, TCP/IP traffic traveling to the protected network via the Internet can reach an intended computer on the internal network. The technique employed requires little or no alteration of the intended internal network device, firewall, proxy server, or security policy configurations, so long as outgoing connections are permitted via at least one protocol, such as, for example, HTTP. The outgoing connections can be made via a proxy server if necessary. Yet, even though the outgoing connection can be limited to one protocol, incoming traffic is not limited to the one protocol and can employ any protocol the Internet and the protected network, and the intended device, are capable of transmitting and/or handling.

Public addressability of the protected network is not required, yielding access to the private, non-unique address space that is not ordinarily routable from clients outside the protected network. Still, the technique preserves network security via several built-in security measures.

[0005] The technique applied by various embodiments of the invention is referred to as "Reverse Proxying," in part because it includes two primary components: a proxy agent, located within the protected network; and an external proxy server, located outside the protected network (for example, on the Internet) at a location reachable by the proxy agent. The external proxy server stores traffic addressed to devices within the protected network until a proxy agent discovers queued traffic intended for the protected network, at which point the external proxy server forwards this traffic to the intended internal network device(s). In turn, the proxy agent forwards any responses it receives from the internal network device(s) back to the external proxy server, which transmits the responses to the intended clients.

[0006] The external proxy server represents clients connecting to the internal (protected) network devices; for example, clients can establish TCP/IP connections to the proxy server and send and receive data to the external proxy server on designated TCP/IP ports that are, in effect, forwarded by the external proxy server to the proxy agent. Likewise, the proxy agent connects to the otherwise inaccessible internal network devices, and sends/transmits and receives data as if it were the client. To a real external client, the external proxy server is the internal network device—the external proxy server thus masquerades as, or "pretends to be," the internal network device. To an internal network device, the proxy agent is the external client—the proxy agent thus masquerades as, or "pretends to be," the client. The link between the external proxy server and the proxy agent is transparent to both the external client and to the internal network device, and is of no concern to them.

[0007] To effect the transparent connection, various embodiments of the invention employ "trickle down polling" to reduce latency and provide highly responsive service without imposing the high network loads that can result from too-frequent polling. In addition, several security measures can be built-in to ensure that it cannot be used to compromise the integrity and privacy of the networks it services, up to the highest standards met by current Internet applications. For example, communication between the proxy agent and the external proxy server can be encrypted using an encryption system, such as the industry standard Secure Sockets Layer (SSL) for HTTP, preventing eavesdropping. Authentication of both the agent and the Server can be enforced by requiring, for example, X.509 certificates of both, or using another authentication technique, such as other "public key" based cryptography systems, and can be verified by a trusted certification authority. The external proxy server also implements a cookie rewriting process, ensuring that all cookies have truly unique identifiers; if a browser should attempt to transmit a cookie to a destination for which it is not intended, the external proxy server will silently drop the cookie from the request. Further, network administrators can be given fine-grained control over the Reverse Proxying system.

[0008] More specifically the present invention relates to a reverse proxy network communication scheme wherein a

proxy agent located inside a protected network is addressable by internal network devices. The proxy agent establishes outgoing network connections on behalf of the internal network devices through a security device, such as a firewall, through which all traffic between the protected network and external networks, such as networks and external network devices on the Internet, must travel. The security device permits at least outgoing connections via at least one predetermined network protocol, such as HTTP.

[0009] An external proxy server outside the protected network is reachable by the proxy agent via outgoing network connections through the security device. The external proxy server is addressable by external network devices, thereby allowing communication between the external network devices and the internal network devices.

BRIEF DESCRIPTION OF THE DRAWINGS

[0010] This disclosure includes the attached Figures, which Figures are summarized as follows:

[0011] FIG. 1 illustrates a typical protected network connected to the Internet.

[0012] FIG. 2A shows a simplified schematic of the connections between a client machine on a protected network and a sever on the Internet.

[0013] FIG. 2B shows a simplified schematic of the connections between a client machine on the Internet and a server on a protected network according to principles of the invention described in this application.

[0014] FIG. 3 shows a more detailed schematic of the connections between client machines and servers on protected networks according to principles of the invention described in this application.

[0015] FIG. 4 depicts two exemplary private networks, to which a web browser is connected, through a reverse proxy server. The two distinct networks have identical private network addresses, and the figure shows how cookies originating from these networks may be confused by the browser.

[0016] FIG. 5 shows an exemplary timeline of an HTTP cookie protocol that can be used in embodiments of the invention where a browser connects to a unique network address space.

[0017] FIG. 6 shows an exemplary timeline of an HTTP cookie protocol that can be used in embodiments of the invention where cookies from duplicate private network address spaces are confused.

DETAILED DESCRIPTION OF THE INVENTION

[0018] In various embodiments of the invention, communication between a device internal to a protected network and a device external to a protected network can be achieved where conventional security devices, such as firewalls and/or proxy servers, would not allow such communication. For example, incoming TCP/IP connections from a network 10, such as the Internet, outside a firewall-protected network 50 to protected/internal devices on the protected network can occur. The technique used in various embodiments requires no alteration of the firewall 20 configuration or existing security policies, provided that the firewall 20 permits

outgoing HTTP connections from the protected/internal device. Incoming connections are not restricted to any particular protocol, such as HTTP, but may be any appropriate networking protocol, including, but not limited to, FTP, gopher, smtp, pop, http, rtsp, and IPX. The outgoing connections are not limited to HTTP, but can be any appropriate protocol the networks, firewall, and/or proxy servers can handle. No alteration of the devices typically connected to a protected network is required, nor does a system deployed according to the principles of the invention require that the protected network 50 be publicly addressable. The technique employed will function unaltered in a private, non-unique address space not ordinarily routable for clients on the Internet 10. Several built-in security measures maintain the privacy of the firewalled network.

[0019] FIG. 1 illustrates a highly secure network configuration with dual firewalls 20, a public "Demilitarized Zone" (DMZ) segment, and a private address space completely inaccessible to outside hosts. Devices and servers for internal use would be hosted on the private segment and would therefore ordinarily be totally isolated from the Internet 10.

[0020] Applying the techniques of various embodiments of the invention, network traffic between external network devices and internal network devices hidden behind the security device 20 is possible even though the protected network uses a private address space. For example, embodiments similar to that shown in FIG. 2B can have TCP/IP network connectivity between an external device and devices hidden behind firewalls 20. The only assumption made is that outgoing connections, such as HTTP connections, are permitted by the existing firewall configurations, possibly through an outgoing proxy server, and by corporate security policies. No alterations are required to:

- [0021] 1. The networked devices.
- [0022] 2. The firewalls used to protect the network.
- [0023] 3. Corporate security policies.
- [0024] 4. The address spaces
- [0025] 5. The clients used to connect to the hidden devices
- [0026] 6. The TCP/IP protocol used by the client and server

[0027] The absence of such alterations can render the processes of the present invention easy and inexpensive to deploy, with substantially no disruption of the existing network, which can be a considerable improvement over existing solutions.

[0028] As illustrated in, for example, FIGS. 2B, and 3-6, "Reverse Proxying" primarily comprises two components: the proxy agent 240 and the external proxy server 250. The proxy agent 240 is located within the protected network 50. It is assumed that this agent has the ability to establish outgoing network connections, such as HTTP connections, possibly through an outgoing HTTP proxy server, to the Internet 10. For the purposes of explaining the operation of embodiments of the invention, particular protocols will be used, but the invention is not limited to the particular protocols used in this example. The external proxy server 250 is located outside the protected network 50, on the Internet 10, at a location reachable by the agent and receives

traffic addressed to internal network devices. The proxy agent 240 periodically polls the external proxy server 250 to check for queued traffic intended for the protected network 50. When the proxy agent 240 discovers traffic intended for internal network devices, it forwards this traffic to the intended recipients. In turn, the proxy agent 240 will forward any responses it receives back to the external proxy server 250, which will transmit the responses to the intended external network device clients. FIG. 3 illustrates an embodiment of this architecture:

[0029] For clients connecting to the hidden (protected) internal network devices, the external proxy server 250 represents those devices and thus masquerades as the internal network devices. In various embodiments of the invention, clients establish TCP/IP connections to the proxy server 250, and send and receive data to the external proxy server 250, on designated TCP/IP ports that are, in effect, forwarded by the external proxy server 250 to the proxy agent 240. Likewise, the proxy agent 240 connects to the otherwise hidden internal network devices, and sends and receives data as if it is the external network device client. Thus, the proxy agent 240 masquerades as the external network device client. The link between the external proxy server 250 and the proxy agent is transparent to both the external network device client and the internal network device, and is of no concern to them.

[0030] As mentioned above, in various embodiments of the invention, connections and data received by the external proxy server 250 are stored for later retrieval by the proxy agent 240. The proxy agent polls the external proxy server 250 at regular intervals, using, for example, an HTTP connection, to discover pending connections and data, and deliver responses from the intended internal network devices. In effect, the TCP/IP traffic between the external network device client and the internal network device is "tunneled" through HTTP in this way, encapsulated in HTTP requests and responses with header information indicating the source and destination IP addresses and the intended ports. To improve efficiency, multiple requests can be multiplexed through the same HTTP connection.

[0031] It is instructive to compare the Reverse Proxying, with traditional "forward" proxying. FIGS. 2A and 2B illustrate the difference between traditional proxying (FIG. 2A) and the reverse proxying employed by embodiments of the invention (FIG. 2B).

[0032] Providing access to private IP addresses is what allows the success and generality of this scheme. The private IP address spaces 50 are not unique across the Internet 10 and many different organizations reuse the same IP address spaces 50. For the IP address spaces 50 and the internal network devices 200 residing therein to be addressable by external network device clients 230, the external proxy server 250 maintains a map between local TCP/IP ports on the proxy server 250 and remote private IP addresses distinguished by the identity of the proxy agent used to access them. Proxy agents publish a list of addresses they can reach to the external proxy server 250, and this list is used by the external proxy server 250 to establish the map between local ports and agents/remote addresses.

[0033] No assumptions need be rendered regarding the network protocol used by the external network device client

to communicate with the internal network device and/or (hidden) server on the protected network. All network traffic, for example TCP/IP traffic, is tunneled by the proxy agent 240 through the exemplary HTTP connection between the proxy agent 240 and the external proxy server 250, and there is generally no need for them to alter this data, with some notable exceptions. Certain protocols can require special treatment, particularly HTTP itself. The use of embedded hyperlinks in HTML pages implies that a client may be redirected by a link to an inaccessible URL hidden behind the security device/firewall 20, away from the external proxy server 250 which enables its access to the hidden network. To prevent or minimize such undesirable redirection, a web browser/external client device 230 can be configured (through standard browser settings) to use the external proxy server 250 as a true HTTP proxy server, using the local port on the server described above. This ensures that all HTTP requests are forwarded intact and uninterpreted to the external proxy server 250, which passes those requests to the proxy agent 240. The agent 240 retrieves the requested URLs, which are directly accessible to it since it is behind the firewall 20.

[0034] The proxy agent 240 is forced to poll the external proxy server 250 for pending traffic because it is assumed that only outgoing HTTP connections are permitted by the network security device 20. This introduces a latency problem, since the polling interval determines the responsiveness of the TCP/IP traffic tunneled over the polled HTTP connection. Latency refers to delays introduced by the time it takes for traffic to travel from an origin to a destination and from the destination back to the origin. Since traffic must be queued by the proxy server until the proxy agent polls it, there is a delay between arrival of the traffic at the proxy server and arrival at the proxy agent, increasing the latency. High latency, delays on the order of tenths of a second or more, between requests and responses can compromise the practical usability of a system employing reverse proxying. Latency can be reduced by a decreased polling interval, but this imposes an increasing network load burden and can be limited by the minimum time required to establish and complete an outgoing HTTP request.

[0035] To reach a suitable compromise between latency reduction and network load, various embodiments of the invention employ "trickle down polling to reduce latency and provide highly responsive service without imposing the high network loads implied by too-frequent polling. The proxy agent 240 connects to the external proxy server 250 to discover pending traffic. If there is nothing pending, the external proxy server 250 returns a slow stream of spurious bytes which are ignored by the proxy agent 240. When the external proxy server 250 receives data from an external network device or client/browser 230, it is immediately transmitted to the proxy agent 240 and the connection is closed to flush any buffering performed by intervening (outgoing) proxy servers. To improve response times, the agent 240 can open several connections to the proxy server 250 to reduce the likelihood that no connections will be open when traffic arrives. The trickling-down of spurious bytes prevents any timeouts on the outgoing HTTP request, which may be enforced by intervening outgoing proxy servers. In this way, highly responsive service is guaranteed since the proxy agent 240 can usually be informed immediately of incoming traffic, removing the undesirable latency between the time that this traffic is queued on the external proxy

server 250 and the time that the proxy agent 240 retrieves it. However, the Internet 10 itself can impose a lower bound on latency since it can determine the time taken to transmit requests and responses, and network protocols used by the Internet, such as TCP/IP, do not provide guaranteed service.

[0036] Several security measures can be built into the invention to ensure that it cannot be used to compromise the integrity and privacy of the networks it services, up to the highest standards met by current Internet applications.

[0037] Communication between the proxy agent 240 and the external proxy server 250 can, for example, be encrypted using an encryption system, such as the industry standard Secure Sockets Layer (SSL) for HTTP, preventing eavesdropping. Authentication of both the agent 240 and the server 250 can be enforced by requiring, for example, X.509 certificates of both, or using another authentication technique, such as other "public key" based cryptography systems, and can be verified by a trusted certification authority. The external proxy server 250 can also implement a cookie rewriting process, such as the exemplary process illustrated in FIGS. 4-6, ensuring that all cookies have truly unique identifiers.

[0038] As shown in FIG. 5, web servers 200 can request that clients 230 (web browsers) maintain state through a mechanism known as "cookies". To effect cookies, servers insert additional headers onto replies to HTTP requests, which specify named "echo" data that the browser should repeat back to the server when accessing certain resources identified in the header. Each data element to be stored and echoed is called a "cookie."

[0039] Following such a cookie protocol, a web browser associates cookies with the Uniform Resource Locators (URLs) to which they were bound by the web server. In normal Internet usage, these URLs are guaranteed to be unique. However, in a reverse proxying situation, in which private network addressing becomes a factor, these URLs are not necessarily unique—this is true whether or not IP addresses or symbolic names are used in the URL, since symbolic domain names need not be unique across private IP spaces. This can create two problems:

[0040] 1. Race conditions. In this situation, the browser overwrites an existing cookie for a URL with the most recent value tied to that URL. There is consequently a race between servers to set the cookie data. A server that associates cookie data with a URL is thus not guaranteed that it will receive the same data back. This can partially or totally disable web servers/applications that rely on correct state data echoed in cookies.

[0041] 2. Privacy violations. In this situation, cookie data associated with a URL can contain private data from a protected network, since servers in such networks can assume that all transmission between themselves and clients is secured. However, the browser could now unwittingly transmit this private data to a wholly different network, since it confuses the non-unique URLs. Servers in the wrong network might therefore gather sensitive data from other private networks, intentionally or unintentionally, in this way. This can be a serious compromise of the network security established by the firewall/private IP space system.

[0042] FIG. 4 illustrates how cookies from different networks can be confused by web browsers. Web clients (browsers) 230 use URLs to uniquely identify resources on the Internet 10. This is both specified by the relevant standards and by common practice. However, by providing access to private/protected networks 50 with not-necessarily-unique URLs, reverse proxying schemes create potential confusion between these URLs. This only becomes an issue, however, when a stored state is associated with a (non-unique) URL(s) and transmitted later as part of requests for other networks, since all current requests are explicitly directed to the proper destinations by the proxy server configuration. This situation is analogous to luggage-handling errors on airline flights, where the incorrect luggage is transported on a flight that is directed to an otherwise-correct destination, due to a non-unique label on the luggage.

[0043] In various embodiments of the invention, a process referred to as "cookie rewriting" eliminates cookie ambiguity. All cookies have names. Typically, proxy servers do not alter any data sent or received by proxy. In various embodiments, the invention makes an exception for cookie names, which are rewritten by the proxy server as they are transmitted back to browsers for storage, to indicate clearly which private network they originate from. The reverse proxying scheme has some way of distinguishing private networks in embodiments of the invention (e.g. by the identity of the agent within those networks which effects firewall traversal) or the proxy server would not function correctly. One way of doing this is to prepend the unique identity of the private network to each cookie name (that is, place the private network identifier at the "front" of the cookie as a "prefix"), which is the implementation used in various embodiments of the invention, though other rewriting methods are possible. The prefix can then be stripped from the cookie when it is transmitted. Cookies passed by the browser with a request which originated from a different network are silently dropped by the proxy server. Thus the external proxy server maintains the privacy of the networks and ensures correct cookie storage and passing by browsers.

[0044] In the situation shown in FIG. 6, a browser first issues an HTTP GET request for the URL `http://someserver`, via the Proxy Server. The browser is configured to use Port A on the Proxy Server, which associates Port A with the private network A. The Proxy Server performs the request on the behalf of the browser (using whatever firewall traversal scheme it supports), and inspects any cookies which the someserver returns in the response. In this case, the cookie xyz with the value s has been set by someserver. The Proxy server rewrites the name of the cookie to `A_xyz` so it is clearly marked as a cookie intended for private network A. Note that the web browser attaches no intrinsic meaning to cookie names, simply echoing them to the URLs they are associated with. The browser receives the HTTP response from the proxy server, and stores the cookie `A_xyz=s`.

[0045] Later the browser is reconfigured to use Port B on the Proxy Server, which associates port B with the private network B. The browser issues an HTTP GET request for the same URL `http://someserver`, sending the cookie `A_xyz=s` with the request. It does so because it has no way of determining that the intended network has changed. The Proxy Server inspects any cookies contained in the request before forwarding it to someserver in the network B. Since

the cookie A_xyz=s is intended for A and not B, it is discarded by the Proxy Server, and the rest of the request is forwarded. As before, the Proxy Server rewrites the names of any cookies contained in the HTTP response, so that xyz=t becomes B_xyz=t. This ensures that, in future, the cookie will not be passed to the network A, or any other network it was not intended for.

[0046] In addition to the above security measures, network administrators can be given fine-grained control over the Reverse Proxying system. For example, administrators can be granted the authority and/or ability to allow or deny entry into their network on a per-session basis by granting a permission, such as a short-lived key; administrators can also be granted the authority and/or ability to completely disable access, or limit it by other criteria.

[0047] The preceding description of the invention is exemplary in nature as it pertains to particular embodiments disclosed and no limitation as to the scope of the claims is intended by the particular choices of embodiments disclosed.

[0048] Other modifications of the present invention may occur to those skilled in the art subsequent to a review of the present application, and these modifications, including equivalents thereof, are intended to be included within the scope of the present invention.

What is claimed is:

1. A reverse proxy network communication scheme comprising:

a proxy agent located inside a protected network addressable by a least one internal network device, the proxy agent establishing outgoing network connections;

a security device through which all traffic between the protected network and external networks must travel, the security device permitting at least outgoing connections via at least one predetermined network protocol;

an external proxy server outside the protected network and reachable by the proxy agent via outgoing network connections through the security device, the external proxy server also being addressable by at least one external network device, thereby allowing communication between the at least one external network device and the at least one internal network device.

2. The scheme of claim 1 wherein the at least one predetermined network protocol is HTTP.

3. The scheme of claim 1 further including an outgoing proxy server in communication with the proxy agent and which the proxy agent uses to establish outgoing connections.

4. The scheme of claim 1 wherein the external proxy server is in communication with at least one other network, receives, and stores data addressed to the at least one internal network device.

5. The scheme of claim 4 wherein the proxy agent polls the external proxy server to check for data addressed to the at least one internal network device.

6. The scheme of claim 5 wherein the proxy agent downloads data addressed to the at least one internal network device from the external proxy server and forwards the data to the at least one internal network device.

7. The scheme of claim 4 wherein the external proxy server ensures proper cookie routing.

8. The scheme of claim 1 wherein the proxy agent forwards outgoing data to the external proxy server, which transmits the data to the at least one external network device.

9. A method of accessing an internal network device on a protected network, the network including a security device, the method comprising:

storing data addressed to the internal network device in an external proxy server;

maintaining a proxy agent on the protected network, the proxy agent executing the step of:

polling the external proxy server for data addressed to the internal network device;

forwarding to the internal network device any data on the external proxy server and addressed to the internal network device; and

forwarding to the external proxy server any data addressed to an external device in communication with the external proxy server.

10. The method of claim 9 further comprising polling the external proxy server at regular intervals.

11. The method of claim 9 further comprising communicating by the internal network device with the external proxy server using a first network protocol and the external network device communicates with the external proxy server using a second network protocol.

12. The method of claim 11 wherein data addressed to the internal network device using the second network protocol is transmitted to the internal device using the first network protocol so that the second network protocol is carried to the internal network device inside the first network protocol.

13. The method of claim 9 further including multiplexing multiple requests from the proxy agent to the external proxy server through the same connection.

14. The method of claim 9 further including maintaining by the external proxy server of maps between local TCP/IP ports of the external proxy server and private IP addresses on the protected network, the maps being distinguished by an identity of the proxy agent used to access them.

15. The method of claim 14 further including publishing by each proxy agent a list of addresses it can reach to the external proxy server, the external proxy server using this list to create a respective map between local ports and proxy agents.

16. The method of claim 14 further including ensuring cookie delivery.

17. The method of claim 9 wherein polling comprises:

connecting to the external proxy server to check for pending traffic;

returning a slow stream of spurious bytes ignored by the proxy agent if there is nothing pending;

immediately transmitting data from the external proxy server to the proxy agent when the external proxy server receives data from a client, thus closing the connection to flush any buffering performed by intervening (outgoing) proxy servers.

18. The method of claim 9 wherein communication between the proxy agent and external proxy server is encrypted.

19. The method of claim 18 wherein the data is encrypted using Secure Sockets Layer (SSL) for HTTP.

20. The method of claim 19 wherein both the proxy agent and the external proxy server require X.509 certificates.

21. The method of claim 9 further comprising rewriting cookies with unique identifiers to prevent inadvertent transmission of private information to an incorrect recipient on the protected network.

22. The method of claim 9 further comprising providing network administrators control over the system including granting administrators the ability to allow and deny entry into the protected network on a per session basis.

23. The method of claim 22 wherein access is conferred by granting a key with a predetermined life span.

* * * * *



US 20030182431A1

(19) **United States**(12) **Patent Application Publication** (10) **Pub. No.: US 2003/0182431 A1****Sturniolo et al.**(43) **Pub. Date:****Sep. 25, 2003**

(54) **METHOD AND APPARATUS FOR
PROVIDING SECURE CONNECTIVITY IN
MOBILE AND OTHER INTERMITTENT
COMPUTING ENVIRONMENTS**

Publication Classification(51) **Int. Cl.⁷** **G06F 15/16**(52) **U.S. Cl.** **709/227; 713/200**

(76) Inventors: **Emil Sturniolo**, Medina, OH (US);
Aaron Stavens, Auburn, WA (US);
Joseph T. Savarese, Edmonds, WA
(US)

Correspondence Address:
Nixon & Vanderhye P.C.
8th Floor
1100 North Glebe Road
Arlington, VA 22201 (US)

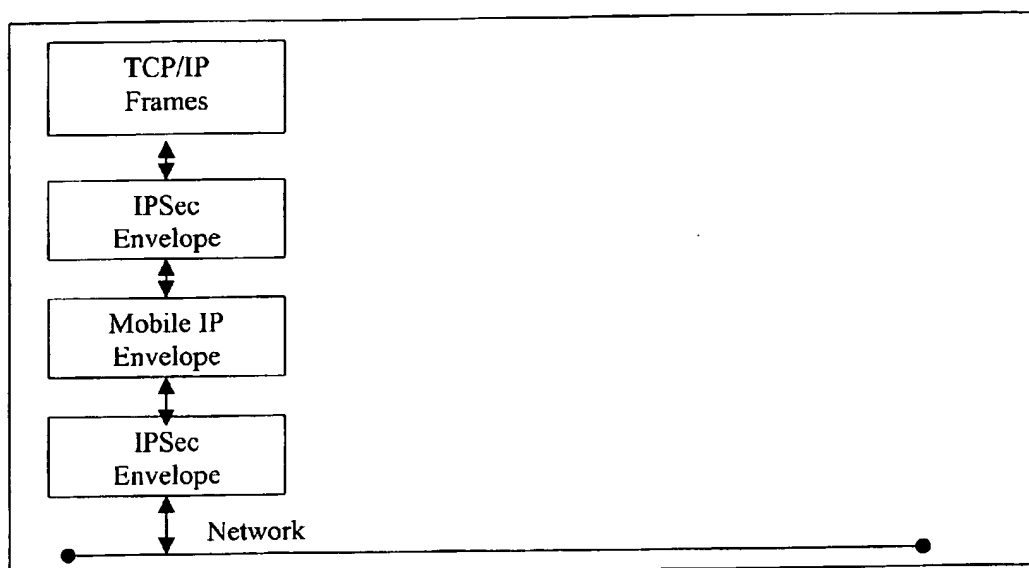
(21) **Appl. No.:** **10/340,833**(22) **Filed:** **Jan. 13, 2003****Related U.S. Application Data**

(63) Continuation of application No. 09/330,310, filed on
Jun. 11, 1999, now Pat. No. 6,546,425.

(60) Provisional application No. 60/347,243, filed on Jan.
14, 2002.

(57) **ABSTRACT**

Method and apparatus for enabling secure connectivity using standards-based Virtual Private Network (VPN) IPSEC algorithms in a mobile and intermittently connected computing environment enhance the current standards based algorithms by allowing migratory devices to automatically (re)establish security sessions as the mobile end system roams across homogeneous or heterogeneous networks while maintaining network application session. The transitions between and among networks occur seamlessly—shielding networked applications from interruptions in connectivity. The applications and/or users need not be aware of these transitions, although intervention is possible. The method does not require modification to existing network infrastructure and/or modification to networked applications.



Enabling Roaming With Mobile IP And IPsec Encapsulation

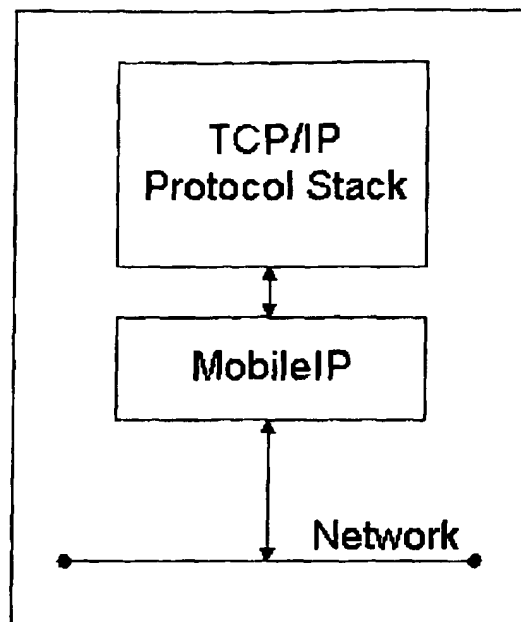


Figure 1 – Example Mobile IP Client Architecture (Prior Art)

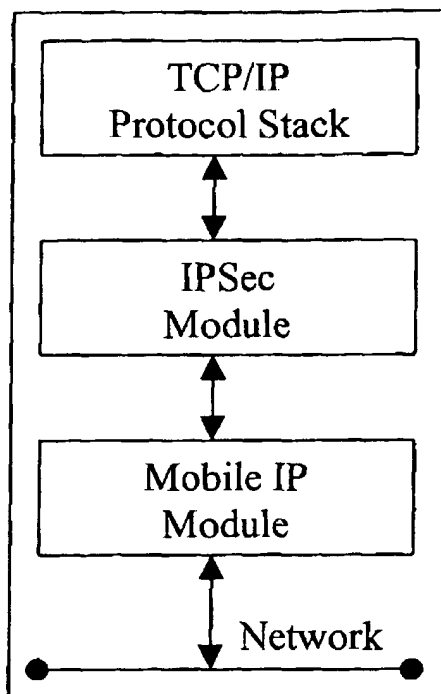


Figure 2: Example IPsec and Mobile IP Architecture (Prior Art)

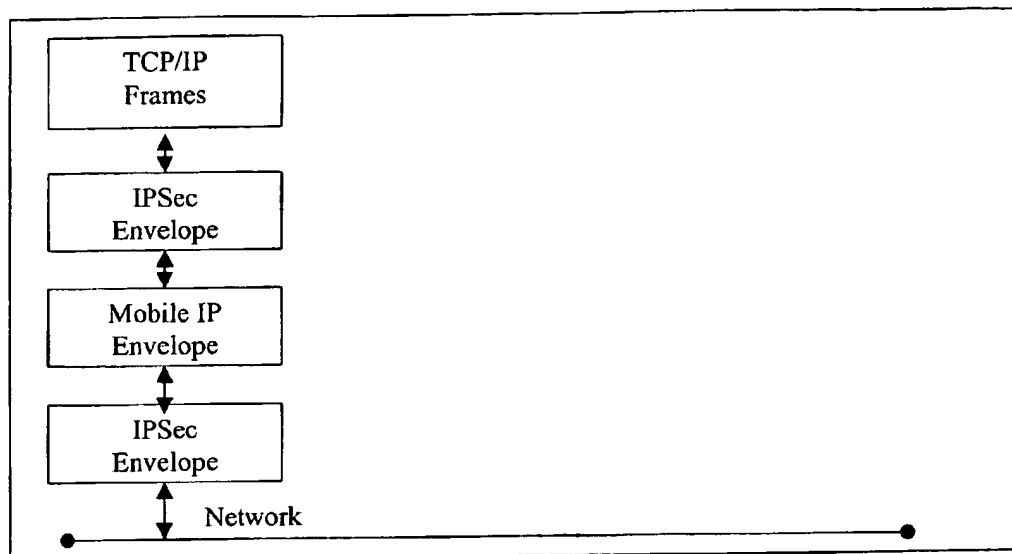


Figure 3: Enabling Roaming With Mobile IP And IPSec Encapsulation

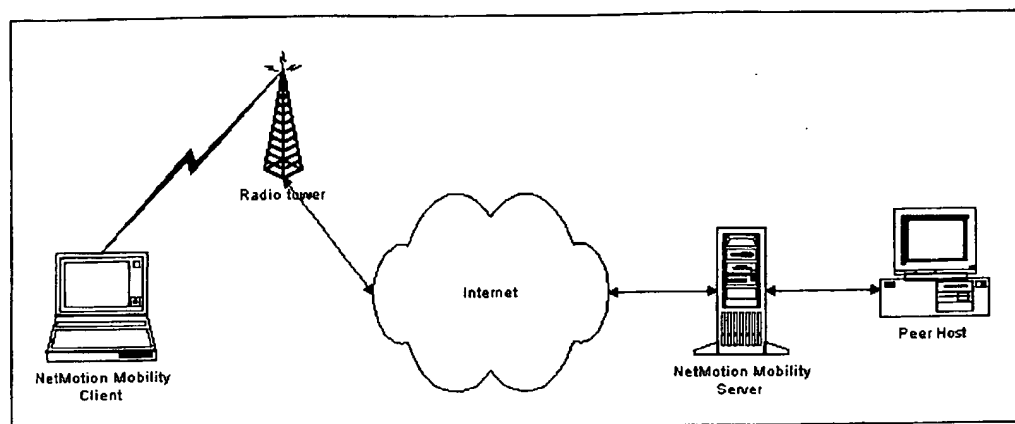


Figure 4 – Example Non-Limiting Secure Mobility Architecture

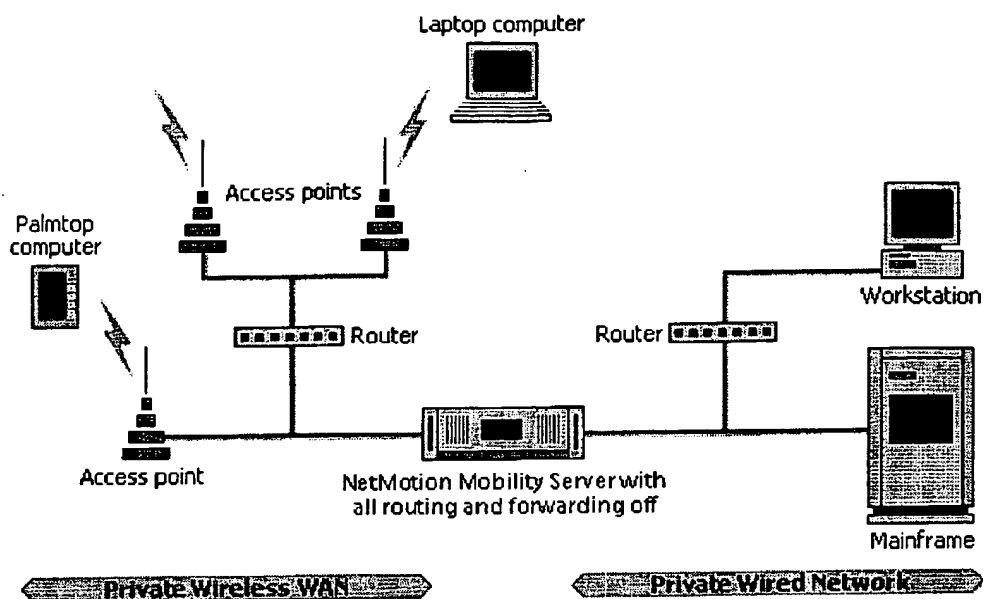


Figure 5A

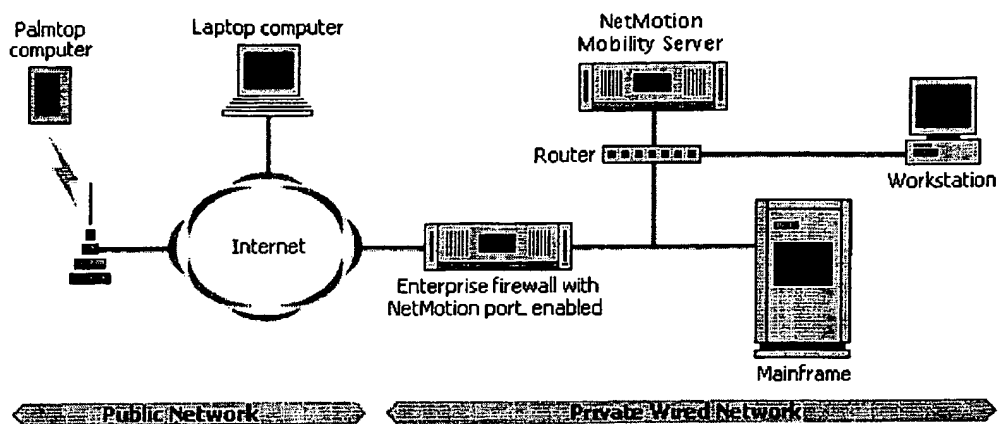


Figure 5B

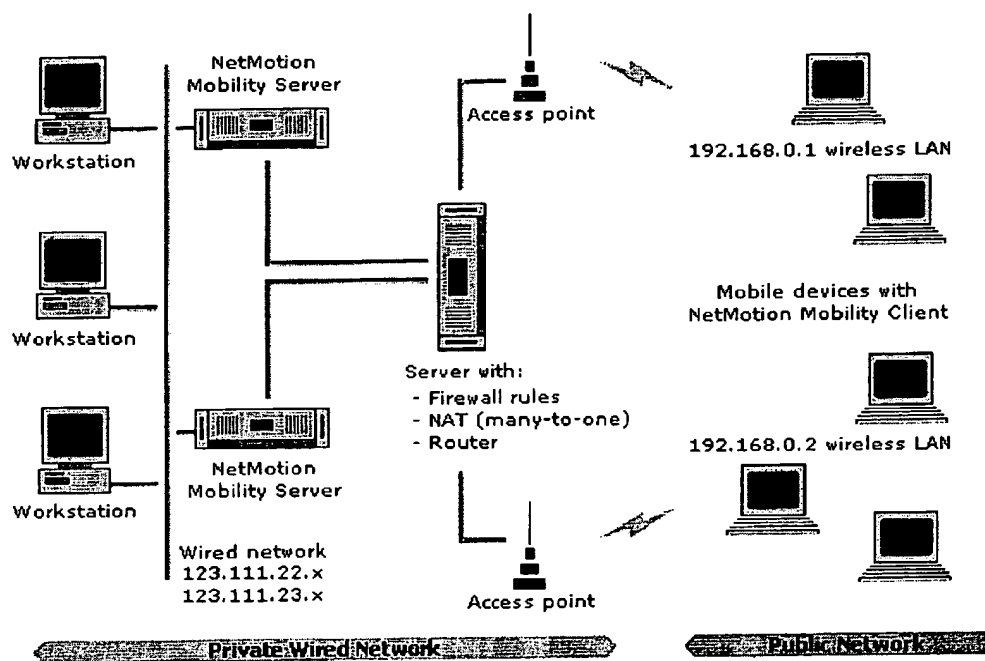


Figure 5C

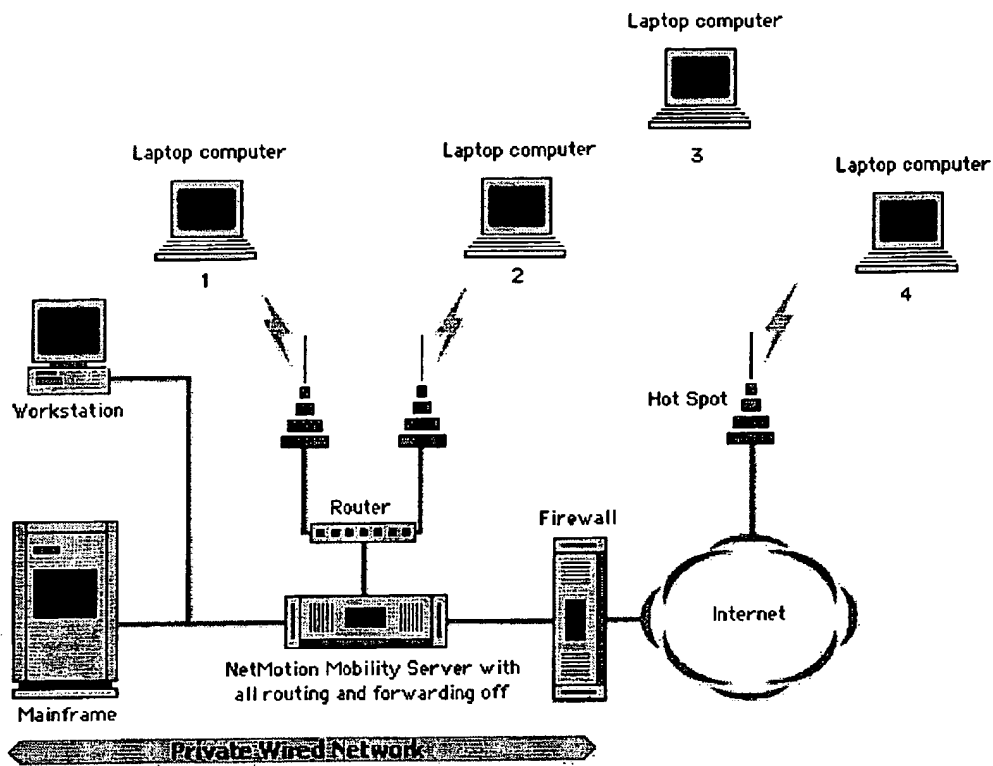


Figure 5D

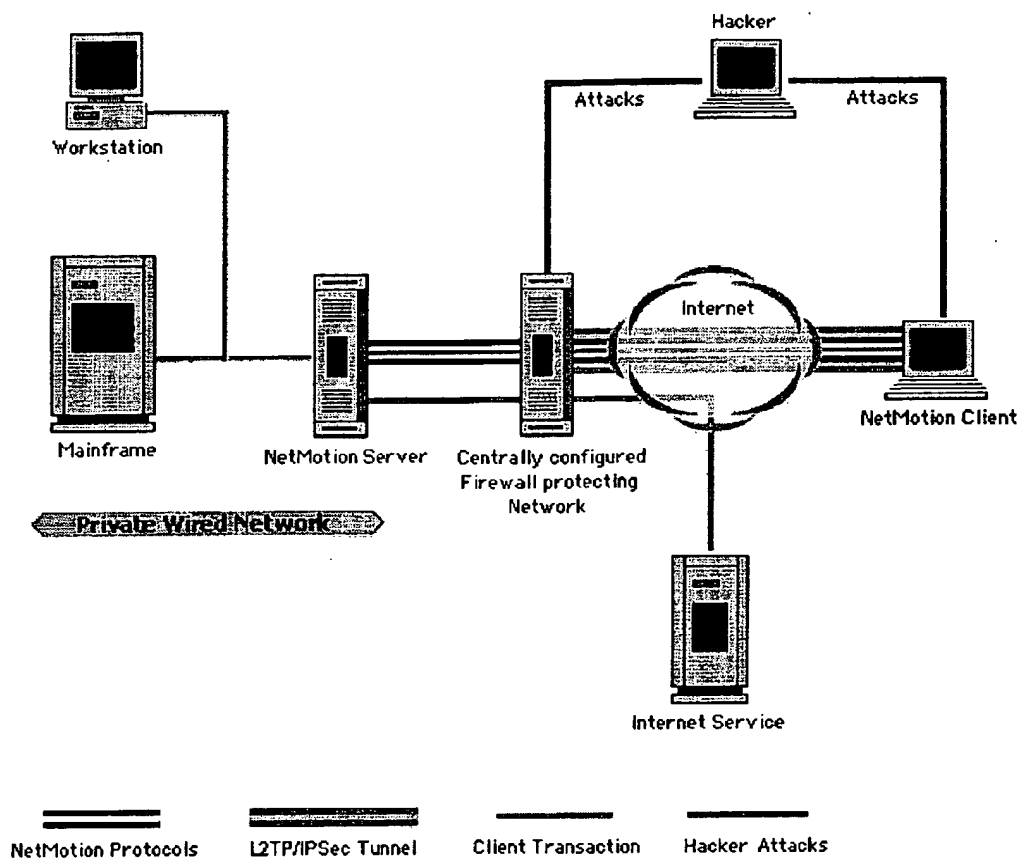


Figure 5E

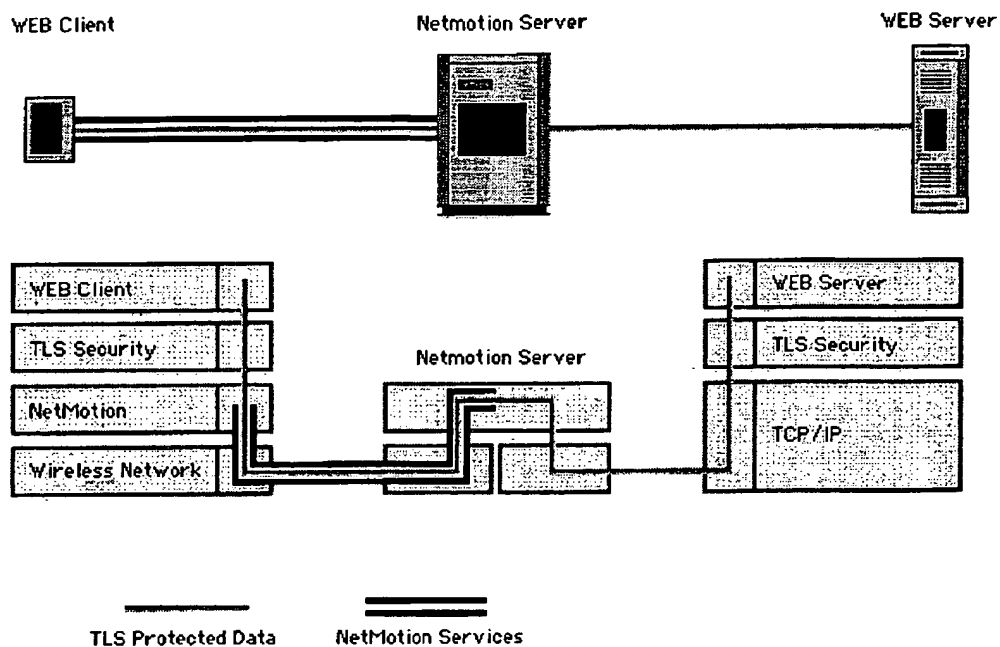


Figure 5F

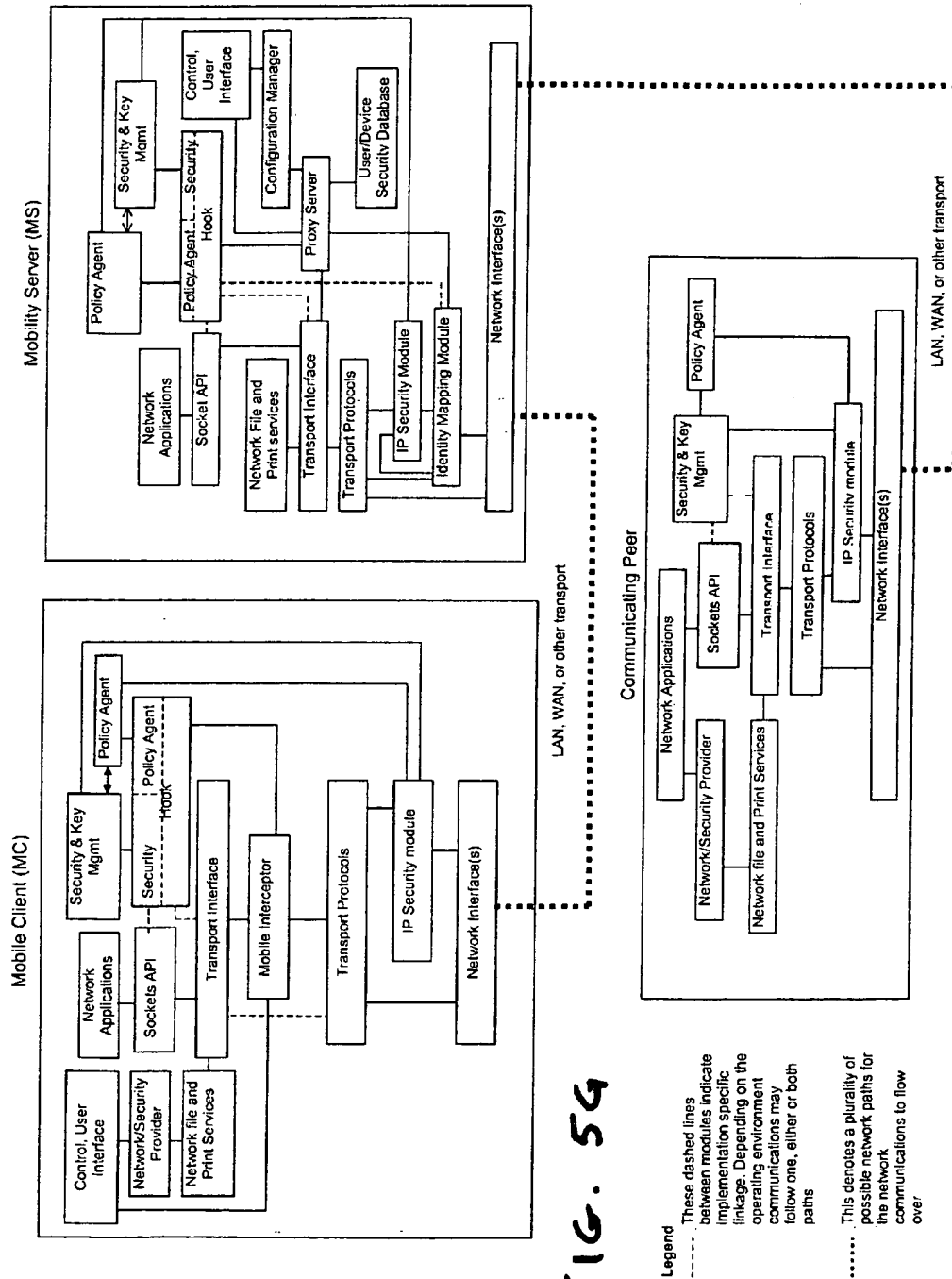


FIG. 54

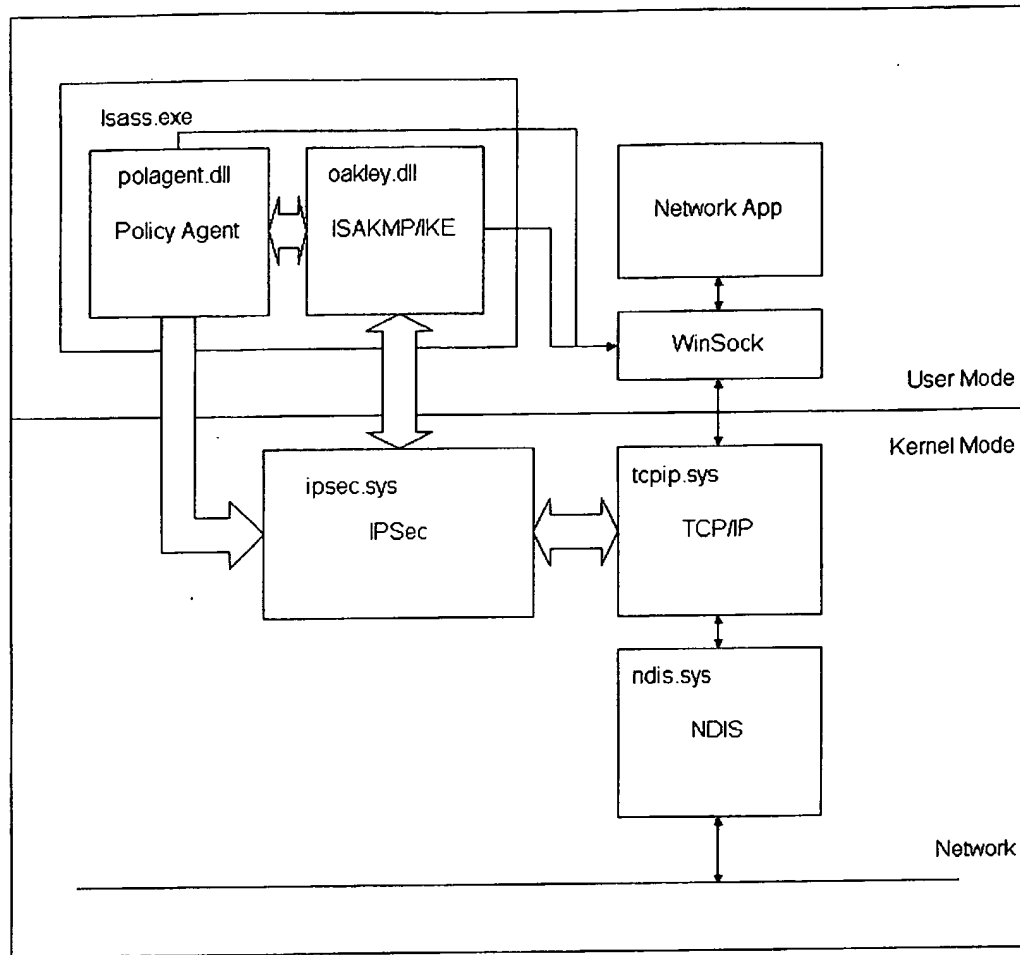


Figure 6 – Example IPSec Operating System Security Architecture
(Prior Art)

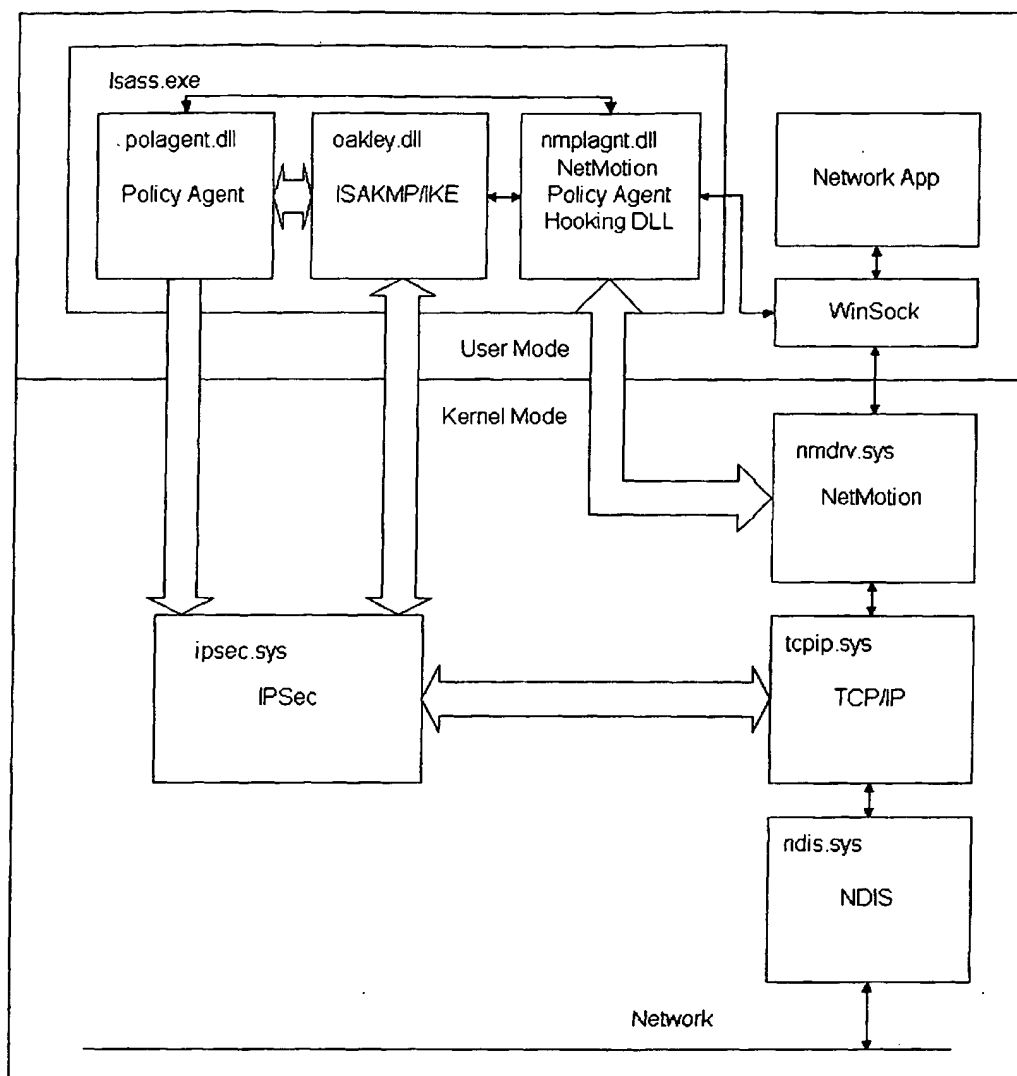


Figure 7 – Example Client Architecture

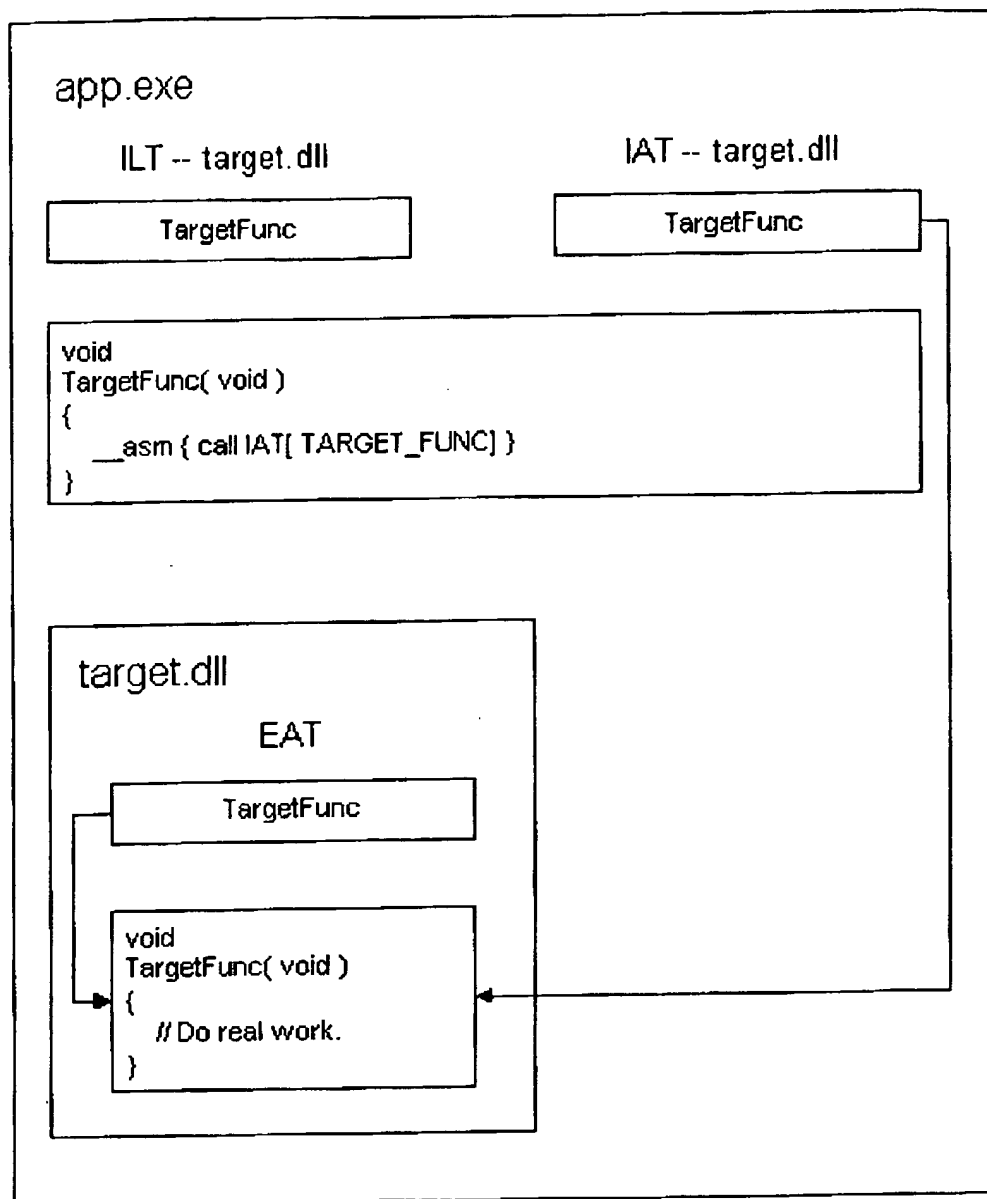


Figure 8 – Example Run-time Linking Sample

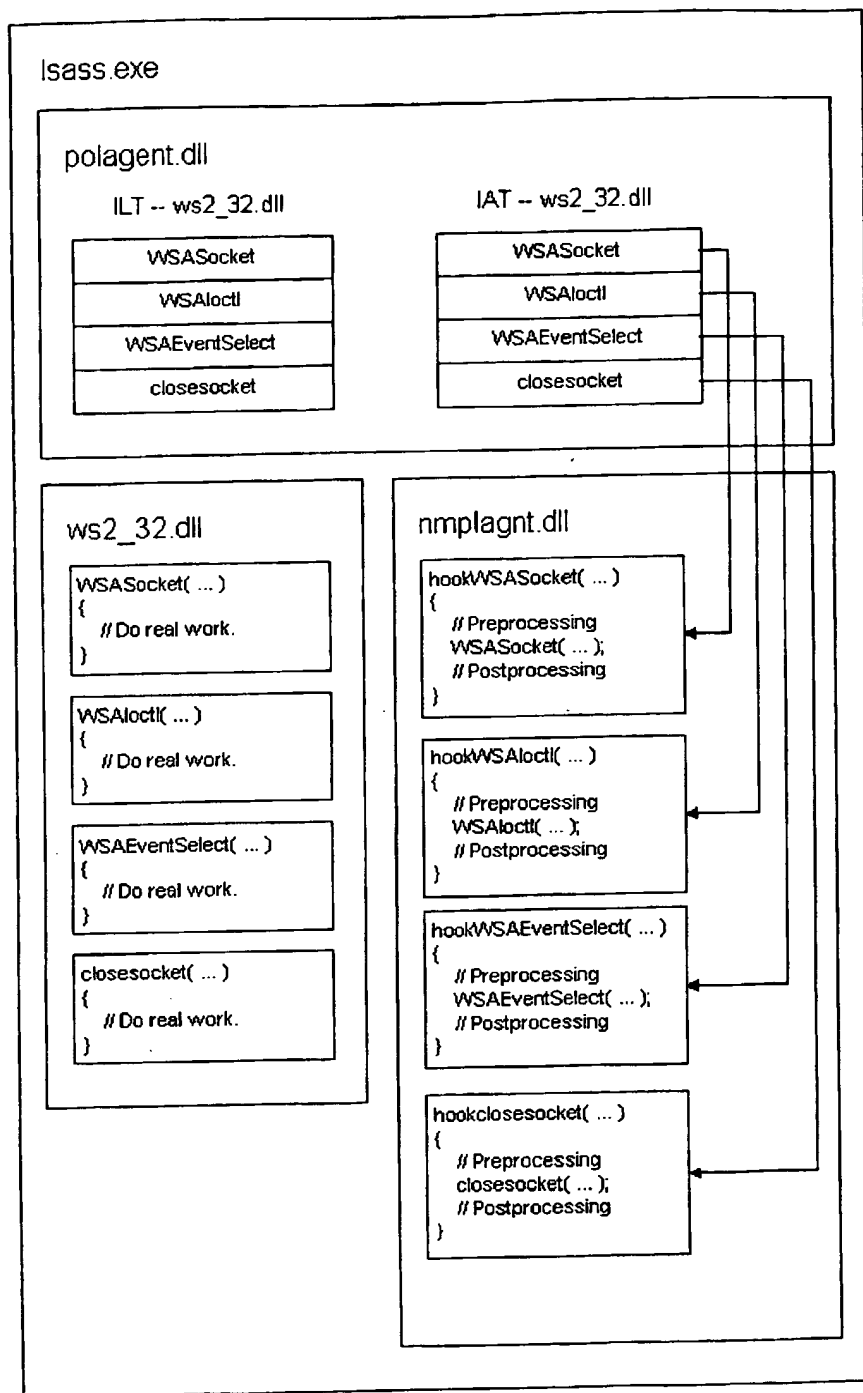


Figure 9 – Example Client Policy Agent Hooking

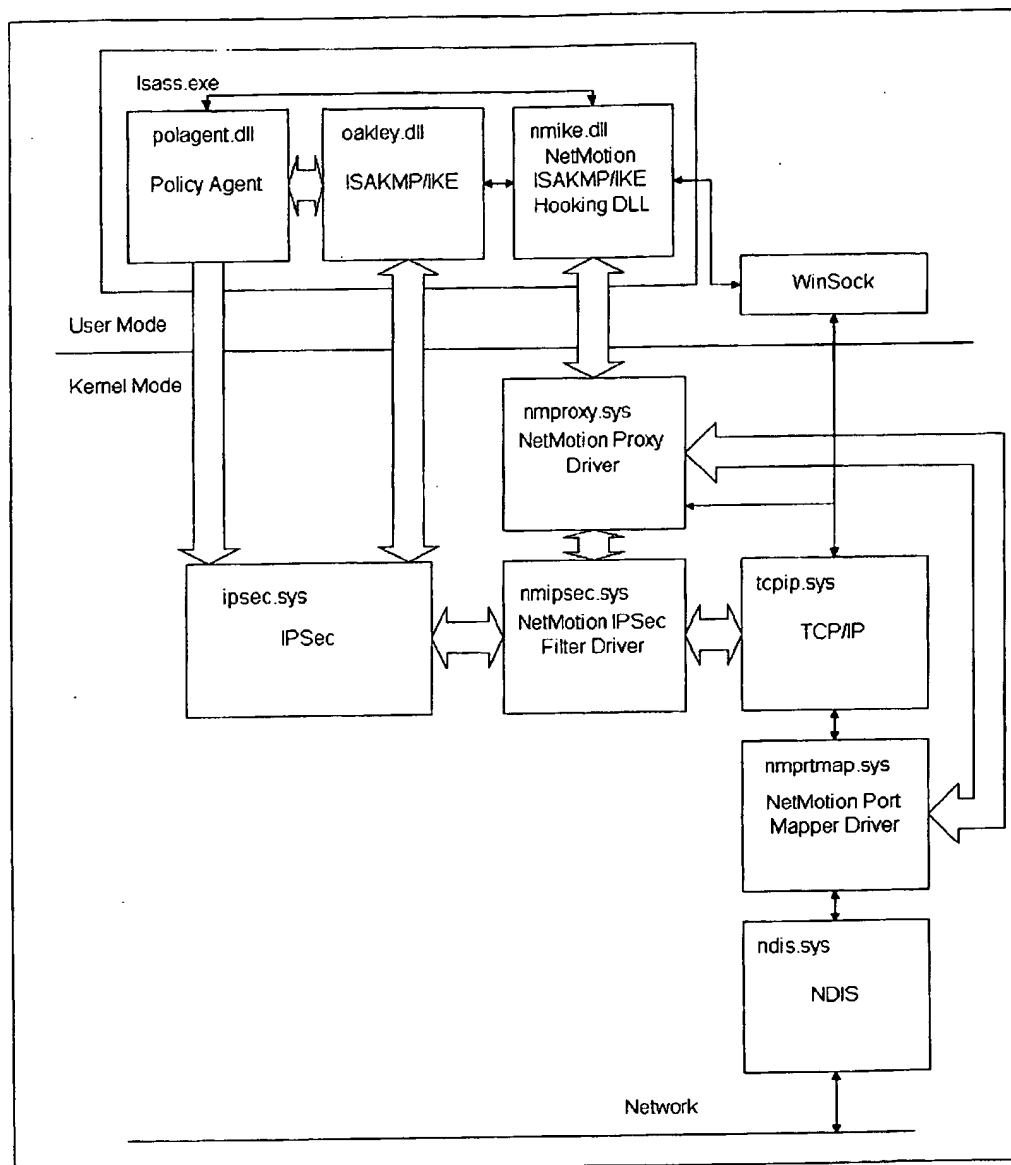


Figure 10 – Example Server Architecture

**METHOD AND APPARATUS FOR PROVIDING
SECURE CONNECTIVITY IN MOBILE AND
OTHER INTERMITTENT COMPUTING
ENVIRONMENTS**

**CROSS-REFERENCE TO RELATED
APPLICATIONS**

[0001] This application claims the benefit of priority from the following copending commonly-assigned related U.S. patent applications:

[0002] U.S. Provisional Application No. 60/347,243, filed Jan. 14, 2002 (Attorney Docket 3978-9);

[0003] U.S. Provisional Application Serial No. 60/274,615 filed Mar. 12, 2001, entitled "Method And Apparatus For Providing Mobile and Other Intermittent Connectivity In A Computing Environment" (Attorney Docket 3978-6);

[0004] U.S. patent application Ser. No. 09/330,310 filed Jun. 11, 1999, entitled "Method And Apparatus For Providing Mobile and Other Intermittent Connectivity In A Computing Environment" (Attorney Docket 3978-3);

[0005] U.S. patent application Ser. No. 09/660,500 filed Sep. 12, 2000, entitled "Method And Apparatus For Providing Mobile and Other Intermittent Connectivity In A Computing Environment" (Attorney Docket 3978-2); and

[0006] PCT International Application Number PCT/US01/28391 filed Sep. 12, 2001, entitled "Method And Apparatus For Providing Mobile And Other Intermittent Connectivity In A Computing Environment" (Attorney Docket 3978-7).

[0007] All of the above-identified documents are incorporated herein by

**STATEMENT REGARDING FEDERALLY
SPONSORED RESEARCH OR DEVELOPMENT**

[0008] Not applicable.

**BACKGROUND AND SUMMARY OF THE
INVENTION**

[0009] Wireless networks have become very popular. Students are accessing course information from the college's computer network while sitting in lecture hall or enjoying the outdoors in the middle of the college campus. Doctors are maintaining computing connectivity with the hospital computer network while making their rounds. Office workers can continue to work on documents and access their email as they move from their office to a conference room. Laptop or PDA users in conference centers, hotels, airports and coffee houses can surf the web and access email and other applications over the Internet. Home users are using wireless networks to eliminate the need to run cables.

[0010] Wireless connectivity provides great flexibility but also presents security risks. Information transmitted through a cable or other wired network is generally secure because one must tap into the cable in order to access the transmission. However, information transmitted wirelessly can be received by anyone with a wireless receiver who is in range.

Security risks may not present much of a problem to students reading course material or to cafe customers surfing the World Wide Web, but they present major concerns to businesses and professionals as well as their clients, customers and patients.

[0011] Generally, wired and wireless computing worlds operate under very different paradigms. The wired world assumes a fixed address and a constant connection with high bandwidth. A wireless environment, in contrast, exhibits intermittent connections and has higher error rates over what is usually a narrower bandwidth. As a result, applications and messaging protocols designed for the wired world don't always work in a wireless environment. However, the wireless expectations of end users are set by the performance and behaviors of their wired networks. Meeting these expectations creates a significant challenge to those who design and develop wireless networking architectures, software and devices.

[0012] Authenticating users and keeping communications confidential are more problematic in a wireless network than they are in a wired network. Wireless networks generally are subject to much greater varieties of attacks (e.g., man-in-the-middle, eavesdropping, "free rides" and wide area imposed threats) and assumptions that often do not apply to wired networks. For example, in modern network topologies such as wireless networks and Internet-based virtual private networks (VPNs), physical boundaries between public and private networks do not exist. In such networks, whether a user has the necessary permissions to access the system can no longer be assumed based on physical location as with a wired network in a secure facility. Additionally, wireless data is often broadcasted on radio frequencies, which can travel beyond the control of an organization, through walls and ceilings and even out into the parking lot or onto the street. The information the network is carrying is therefore susceptible to eavesdropping. Imagine if vital hospital patient information could be intercepted or even altered by an unauthorized person using a laptop computer in the hospital lobby, or if a corporate spy could learn his competitor's secrets by intercepting wireless transmissions from an office on the floor above or from a car in the parking lot. While tapping into a wired network cable in a secure facility is possible, the chances of this actually happening are less likely than interception of radio transmissions from a wireless network. Further security threats and problems must be faced when users wish to use any of the ever-increasing variety of public wireless networks to access sensitive data and applications.

[0013] Many of the open standards that make it possible for wireless network hardware vendors to create interoperable systems provide some form of security protection. For example, the IEEE 802.11b "Wi-Fi" standard has been widely implemented to provide wireless connectivity for all sorts of computing devices. It provides an optional Wired Equivalent Privacy ("WEP") functionality that has been widely implemented. Various additional wireless related standards attempt to address security problems in wireless networks, including for example:

[0014] Wireless Application Protocol (WAP) and the associated Wired Transport Layer Security (WTLS); and

[0015] Mobile IP.

[0016] However, as explained below and as recognized throughout the industry, so far these standards have not provided a complete, easy-to-implement transparent security solution for mobile computing devices that roam between different networks or subnetworks.

[0017] WAP generally is designed to transmit data over low-bandwidth wireless networks to devices like mobile telephones, pagers, PDA's, and the like. The Wired Transport Layer Security (WTLS) protocol in WAP provides privacy, data integrity and authentication between WAP-based applications. A WAP gateway converts between the WAP protocol and standard web and/or Internet protocols such as HTTP and TCP/IP, and WTLS is used to create a secure, encrypted pipe. One issue with this model is that once the intermediate WAP gateway decrypts the data, it is available in clear text form—presenting an opportunity for the end-to-end security of the system to be compromised. Additionally, WAP has typically not been implemented for high-bandwidth scenarios such as wireless local area network personal computer connectivity.

[0018] WEP (Wired Equivalent Privacy) has the goal of providing a level of privacy that is equivalent to that of an unsecured wired local area network. WEP is an optional part of the IEEE 802.11 standard, but many hardware vendors have implemented WEP. WEP provides some degree of authentication and confidentiality, but also has some drawbacks and limitations.

[0019] To provide authentication and confidentiality, WEP generally relies on a default set of encryption keys that are shared between wireless devices (e.g., laptop computers with wireless LAN adapters) and wireless access points. Using WEP, a client with the correct encryption key can “unlock” the network and communicate with any access point on the wireless network; without the right key, however, the network rejects the link-level connection request. If they are configured to do so, WEP-enabled wireless devices and access points will also encrypt data before transmitting it, and an integrity check ensures that packets are not modified in transit. Without the correct key, the transmitted data cannot be decrypted—preventing other wireless devices from eavesdropping.

[0020] WEP is generally effective to protect the wireless link itself although some industry analysts have questioned the strength of the encryption that WEP currently uses. However, a major limitation of WEP is that the protection it offers does not extend beyond the wireless link itself. WEP generally offers no end-to-end protection once the data has been received by a wireless access point and needs to be forwarded to some other network destination. When data reaches the network access point or gateway, it is unencrypted and unprotected. Some additional security solution must generally be used to provide end-to-end authentication and privacy.

[0021] Mobile IP is another standard that attempts to solve some of the problems of wireless and other intermittently-connected networks. Generally, Mobile IP is a standards based algorithm that enables a mobile device to migrate its

network point of attachment across homogeneous and heterogeneous network environments. Briefly, this Internet Standard specifies protocol enhancements that allow routing of Internet Protocol (IP) datagrams (e.g., messages) to mobile nodes in the Internet. See for example Perkins, C., “IP Mobility Support”, RFC 2002, October 1996.

[0022] Mobile IP contemplates that each mobile node is always identified by its home address, regardless of its current point of attachment to the Internet. While situated away from its home, a mobile node is also associated with a “care-of” address, which provides information about its current point of attachment to the Internet. The protocol provides for registering the “care-of” address with a home agent. The home agent sends datagrams destined for the mobile node through a “tunnel” to the “care-of” address. After arriving at the end of the “tunnel,” each datagram is then delivered to the mobile node.

[0023] While Mobile IP provides useful techniques for remote connectivity, it is not yet widely deployed/implemented. This seems to be due to a variety of factors—at least one of which is that there continues to be some unsolved problems or areas where the Mobile IP standard is lacking and further enhancement or improvement would be desirable. For example, even though security is now fairly widely recognized as being a very important aspect of mobile networking, the security components of Mobile IP are still mostly directed to a limited array of security problems such as redirection attacks.

[0024] Redirection attacks are a very real threat in any mobility system. For example, a redirection attack can occur when a malicious node gives false information to a home agent in a Mobile IP network (e.g., sometimes by simply replaying a previous message). This is similar to someone filing a false “change of address” form with the Post Office so that all your mail goes to someone else’s mailbox. The home agent is informed that the mobile node has a new “care-of” address. However, in reality, this new “care-of” address is controlled by the malicious node. After this false registration occurs, all IP datagrams addressed to the mobile node are redirected to the malicious node.

[0025] While Mobile IP provides a mechanism to prevent redirection attacks, there are other significant security threats that need to be addressed before an enterprise can feel comfortable with the security of their wireless network solution. For example, Mobile IP generally does not provide a comprehensive security solution including mobile computing capabilities such as:

[0026] Session resilience/persistence

[0027] Policy management

[0028] Distributed firewall functionality

[0029] Location based services

[0030] Power management

[0031] Other capabilities.

[0032] While much security work has been done by the Internet community to date in the Mobile IP and other contexts, better solutions are still possible and desirable. In particular, there continues to be a need to provide an easy-to-use, comprehensive mobility solution for enterprises and other organizations who wish to add end-to-end security

to existing and new infrastructures that make extensive use of existing conventional technology and standards and which support mobility including roaming transparently to applications that may not be "mobile-aware." Some solutions exist, but many of them require changes to existing infrastructure that can be difficult to implement and maintain.

[0033] For example, in terms of the current implementations that do exist, Mobile IP is sometimes implemented as a "bump" in the TCP/IP protocol stack to replace components of the existing operating system environment. An example of such an architecture is shown in prior art FIG. 1. In the exemplary illustrative prior art arrangement shown, a Mobile IP module sits below the regular TCP/IP protocol stack components and manages the transitions from one network to another. Generally, using such solution, additions or modifications to existing core network infrastructure entities are needed to facilitate the behavior of nomadic or migratory computing. The need for such modifications makes widespread implementation difficult and causes problems in terms of maintainability and compatibility.

[0034] Another common security solution that enterprises have gravitated toward is something called a Virtual Private Network (VPN). VPNs are common on both wired and wireless networks. Generally, they connect network components and resources through a secure protocol tunnel so that devices connected to separate networks appear to share a common, private backbone. VPN's accomplish this by allowing the user to "tunnel" through the wireless network or other public network in such a way that the "tunnel" participants enjoy at least the same level of confidentiality and features as when they are attached to a private wired network. Before a "tunnel" can be established, cryptographic methods are used to establish and authenticate the identity of the tunnel participants. For the duration of the VPN connection, information traversing the tunnel can be encrypted to provide privacy.

[0035] VPN's provide an end-to-end security overlay for two nodes communicating over an insecure network or networks. VPN functionality at each node supplies additional authentication and privacy in case other network security is breached or does not exist. VPN's have been widely adopted in a variety of network contexts such as for example allowing a user to connect to his or her office local area network via an insecure home Internet connection. Such solutions can offer strong encryption such as the AES (Advanced Encryption Standard), compression, and link optimizations to reduce protocol chattiness. However, many or most VPNs do not let users roam between subnets or networks without "breaking" the secure tunnel. Also, many or most VPNs do not permit transport, security and application sessions to remain established during roaming. Another potential stumbling block is conventional operating systems—not all of which are compatible with the protection of existing wireless VPNs.

[0036] To address some of the roaming issue, as previously mentioned, standards efforts have defined Mobile IP. However, Mobile IP, for example, operates at the network layer and therefore does not generally provide for session persistence/resilience. If the mobile node is out of range or suspended for a reasonably short period of time, it is likely that established network sessions will be dropped. This can

present severe problems in terms of usability and productivity. Session persistence is desirable since it lets the user keep the established session and VPN tunnel connected—even if a coverage hole is entered during an application transaction. Industry analysts and the Wireless Ethernet Compatibility Alliance recommend that enterprises deploy VPN technology, which directly addresses the security problem, and also provides advanced features like network and subnet roaming, session persistence for intermittent connections, and battery life management for mobile devices. However, VPN solutions should desirably support standard security encryption algorithms and wireless optimizations suitable for today's smaller wireless devices, and should desirably also require no or minimal modification to existing infrastructure.

[0037] One standards-based security architecture and protocol approach that has been adopted for providing end-to-end secure communications is called "Internet Security Protocol" ("IPSec"). IPSec is a collection of open standards developed by the Internet Engineering Task Force (IETF) to secure communications over public and private networks. See for example:

[0038] RFC 1827 "IP Encapsulating Security Payload (ESP)" R. Atkinson (August 1995);

[0039] RFC 1826 "IP Authentication Header" R. Atkinson. (August 1995); and

[0040] RFC 1825 "Security Architecture for the Internet Protocol" R. Atkinson (August 1995).

[0041] Briefly, IPSec is a framework for ensuring private, secure communications over Internet Protocol (IP) networks, through the use of cryptographic security services. The IPSec suite of cryptography-based protection services and security protocols provides computer-level user and message authentication, as well as data encryption, data integrity checks, and message confidentiality. IPSec capabilities include cryptographic key exchange and management, message header authentication, hash message authentication, an encapsulating security payload protocol, Triple Data Encryption, the Advanced Encryption Standard, and other features. In more detail, IPSec provides a transport mode that encrypts message payload, and also provides a tunnel mode that encrypts the payload, the header and the routing information for each message. To reduce overhead, IPSec uses policy-based administration. IPSec policies, rather than application programming interfaces (APIs), are used to configure IPSec security services. The policies provide variable levels of protection for most traffic types in most existing networks. One can configure IPSec policies to meet the security requirements of a computer, application, organizational unit, domain, site, or global enterprise based on IP address and/or port number.

[0042] IPSec is commonly used in firewalls, authentication products and VPNs. Additionally, Microsoft has implemented IPSec as part of its Windows 2000 and Windows XP operating system. IPSec's tunnel mode is especially useful in creating secure end-to-end VPNs. IPSec VPNs based on public key cryptography provide secure end-to-end message authentication and privacy. IPSec endpoints act as databases that manage and distribute cryptographic keys and security associations. Properly implemented, IPSec can provide private channels for exchanging vulnerable data such as email,

file downloads, news feeds, medical records, multimedia, or any other type of information.

[0043] One might initially expect that it should be relatively straightforward to add a security algorithm such as the standards-based IPSec security algorithm to Mobile IP or other mobility protocol. For example, layering each of the entities in the fashion such as that shown in prior art FIG. 2 would seem to allow for security in an environment where the mobile node's IP address never needs to change. Thus, the IPSec security association between the mobile node and its ultimate peer could be preserved across network segment boundaries, and end-to-end security would also be preserved. However, combining the Mobile IP and IPSec algorithms in this manner can present its own set of problems.

[0044] For example, when the mobile node has roamed to a foreign network and is communicating with its ultimate peer, it is possible that packets generated by the mobile node may be discarded by a policy enforcement entity such as a firewall. This can be due to common practice known as ingress filtering rules. Many firewalls discard packets generated by mobile nodes using their home addresses (internal network identity) and received on an externally facing network interface in defense of the network. This discarding process is intended to protect the network secured by the firewall from being attacked. Ingress filtering has the effect of forcing the tunneling of Mobile IP frames in both directions. See for example RFC 2356 Sun's SKIP Firewall Traversal for Mobile IP. G. Montenegro, V. Gupta. (June 1998).

[0045] Additionally, it is becoming general practice in the industry to require that an IPSec security session be established between the foreign agent and the externally facing policy enforcement equipment (e.g. firewall) before allowing packets to traverse between the external and internal network interconnection (a.k.a. VPN). If the foreign agent is co-located with the mobile node, this can become a cumbersome operation. As exemplary FIG. 3 depicts, yet another level of network protocol enveloping could be used to meet possibly required security policies to allow network traffic to flow between the mobile node to the foreign agent through the policy enforcement equipment (e.g. firewall) to the home agent and then to the other communications end point (i.e. ultimate peer). However, this adds substantial additional overhead due to the additional encapsulation. Furthermore, if the foreign agent entity is not co-located with the mobile node (or up to policy restrictions on the newly attached network), a specific foreign agent may need to be used for these communications, and credential information must somehow be shared between the foreign agent and the terminus of the first (outer) IPSec session. A drawback to this methodology is that it can increase the security risk by sharing credential information with a network entity that may not be directly under user or corporate administrative control.

[0046] What is needed is a solution to these problems providing security, network roaming, and session persistence over conventional information communications networks including but not limited to standard IP based networks without requiring modification to existing network applications. Additionally, it would be useful if such a solution did not require the deployment of Mobile IP or any additional infrastructure such as a foreign agent when vis-

iting a remote network, and the functionality can be transparent to networked applications so they do not need to be modified either.

[0047] This invention solves this problem by transparently providing secure, persistent, roamable IP-based communications using conventional technologies such as IPSec, Microsoft or other operating system security functionality while avoiding the commonly experienced ingress filtering problems. And unlike at least some implementations of Mobile IP, few if any changes are necessary to the underlying network infrastructure.

[0048] Generally, one preferred exemplary non-limiting embodiment provides Mobility Client (MC) functionality that virtualizes the underlying network. Applications running on the mobility client see at least one consistent virtual network identity (e.g. IP address). When an application on the mobility client makes a network request, the mobility client intercepts the request and marshals the request to a Mobility Server (MS) that supports security such as IPSEC. The mobility server unwraps the request and places it on the network as though the server were the client—thus acting as a proxy for the client.

[0049] The reverse also occurs in the exemplary embodiment. When a peer host sends a packet to the mobility client's virtual network identity, the packet is first received by the mobility server and is then transferred to the mobility client. The mobility server maintains a stable point of communication for the peer hosts while the mobility client is free to roam among networks as well as suspend or roam out of range of any network. When the mobility client is out of range, the mobility server keeps the mobility client's sessions alive and queues requests for the mobility client. When the mobility client is once again reachable, the mobility server and client transfer any queued data and communication can resume where it left off.

[0050] Preferred exemplary non-limiting implementations thus offer wireless optimizations and network and application session persistence in the context of a secure VPN or other connection. Wireless optimizations allow data to be transmitted as efficiently as possible to make maximal use of existing bandwidth. For example, the system can be used to switch automatically to the fastest bandwidth network connection when multiple connections (Wi-Fi and GPRS, for example) are active. Network session persistence means that users don't have to repeat the login process when they move from one IP subnet to another, or when they go out of range of the network and return. Exemplary implementations automatically re-authenticate the connection every time users roam, without need for user intervention. Application session persistence means that standard network applications remain connected to their peers, preventing the loss of valuable user time and data. Such optimizations and persistence is provided in the context of a security architecture providing end-to-end security for authentication and privacy.

[0051] In one illustrative embodiment, before data is transported between the network and a mobility client, the network ensures that the end user has the required permissions. A user establishes her identity by logging in to the mobility client using a conventional (e.g., Windows) domain user name and password. Using the conventional domain credentials allows for a single sign-on process and requires

no additional authentication tables or other infrastructure additions. Single sign-on also gives users access to other domain resources such as file system shares. Once a user has been authenticated, a communications path is established for transporting application data. Any number of different protocols (e.g., Common Internet File System, Radius, other) can be used for user authentication. Using certain of these protocols, a mobility server can act as a Network Access Server to secure an initial access negotiation which establishes the user's user name and password using conventional protocols such as EAP-MD5, LEAP, or other protocol. Unlike some wireless protocols, such authentication in the exemplary non-limiting implementations provides user-specific passwords that can be used for policy management allowing access and resource allocation on a user basis.

[0052] Significantly, exemplary non-limiting implementations can be easily integrated with IPSEC or other security features in conventional operating systems such as for example Windows NT and Windows 2000. This allows access to conventional VPN and/or other proven-secure connection technology. IPsec policies can be assigned through the group policy feature of Active Directory, for example. This allows IPsec policy to be assigned at the domain or organizational level—reducing the administrative overhead of configuring each computer individually. An on-demand security negotiation and automatic key management service can also be provided using the conventional IETF-defined Internet Key Exchange (IKE) as specified in Internet RFC 2409. Such exemplary implementations can provide IETF standards-based authentication methods to establish trust relationships between computers using public key cryptography based certificates and/or passwords such as preshared keys. Integration with conventional standards-based security features such as public key infrastructure gives access to a variety of security solutions including secure mail, secure web sites, secure web communications, smart card logon processes, IPsec client authentication, and others.

[0053] Illustrative exemplary embodiments can be cognizant of changes in network identity, and can selectively manage transition in network connectivity, possibly resulting in the termination and/or (re)instantiation of IPsec security sessions between communicating entities over at least one of a plurality of network interfaces. Exemplary illustrative embodiments also provide for the central management, distribution, and/or execution of policy rules for the establishment and/or termination of IP security sessions as well as other parameters governing the behavior for granting, denying and/or delaying the consumption of network resources.

[0054] Illustrative non-limiting advantageous features include:

[0055] Roamable IPsec allows IPsec tunnel to automatically roam with mobile computing devices wherever they go—based on recognized IPsec security standard. Roamable IPsec enables seamless roaming across any physical or electronic boundary with the authentication, integrity and encryption of IPsec, to provide a standards-based solution allowing mobile and remote users with VPN-level security and encryption in an IPsec tunnel that seamlessly roams with wireless users wherever they go and however they access their enterprise data.

[0056] Detecting when a change in network point of attachment, an interruption of network connectivity, a roam to a different network or other subnetworks, a mobile client's identify, or other discontinuity has occurred on the mobile client and (re)instantiating an IP Security session while maintaining network application sessions—all in a manner that is transparent to the networked application.

[0057] Transparently and selectively injecting computer instructions and redirecting the execution path of at least one software or other component based for example on process name to achieve additional level(s) of functionality while maintaining binary compatibility with operating system components, transport protocol engines, and/or applications.

[0058] Selectively but transparently virtualizing at least one network interface for applications and operating system components—shielding them from the characteristics of mobile computing while allowing other components to remain cognizant of interruptions in connectivity and changes in network point of attachment.

[0059] Selectively virtualizing at least one network interface for network applications and operating system components thus shielding them from adverse events that may disturb communications such as changes in network point of attachment and/or periods of disconnectedness.

[0060] Allowing the establishment of multiple IP Security sessions over one or more network interfaces associated with at least one network point of attachment and allowing network application communications to simultaneously flow over any or all of the multiple IP security sessions and correctly multiplex/demultiplex these distributed communication flows into corresponding higher layer communications sessions.

[0061] Applying policy rules to selectively allow, deny, and/or delay the flow of network communications over at least one of a plurality of IP Security sessions.

[0062] Centrally managing and/or distributing policy regarding the establishment of IP Security sessions from a central authority.

[0063] An "Add session" concept—during the proxying of communications for a mobile client, the mobility server can instantiate at least one of a possible plurality of IP Security sessions between a mobility server and an ultimate peer on behalf of a mobility client.

[0064] Establishing and maintaining IP Security sessions between the Mobility Server and ultimate communications peer, even during periods when the mobility client is unreachable.

[0065] Automatically terminating IP Security sessions between the mobility server and ultimate communications peer, based on, but not limited to link inactivity, application session inactivity, or termination of a communications end point.

- [0066] Associating at least one IP security session between the mobility server and ultimate peer and mobility client and mobility server regardless of the current mobility client network identities.
- [0067] Transparently injecting computer instructions and redirecting the execution path to achieve additional level(s) of functionality while maintaining binary compatibility with operating system components, transport protocol engines, and applications.
- [0068] Allowing establishment of at least one of a plurality of IP security sessions over a plurality of network interfaces associated with at least one network point of attachment and allowing network application communications to simultaneously flow over at least one of a plurality of IP Security sessions and correctly multiplex/demultiplex these distributed communication flows back into corresponding higher layer communications sessions.
- [0069] Selectively virtualizing at least one network interface for network applications and operating system components thus shielding them any adverse events that may disrupt communications such as changes in network point of attachment and/or periods of disconnectedness.
- [0070] Centrally managing and distributing policy rules regarding the establishment and/or termination of IP security sessions for mobile clients and/or mobility servers from a central authority.
- [0071] A mobility security solution that starts at the mobile device and provides both secure user authentication and, when needed, secure data encryption.
- [0072] A mobility security solution that voids the need for single-vendor solutions not based on industry-wide, open and other standards.
- [0073] Secure VPN that is extendable to a variety of different public data networks having different configurations (e.g., Wi-Fi network hotspot, wide-area wireless solutions such as CDPD or GPRS, etc.) dynamically controllable by the network administrator
- [0074] A mobility security solution that works with a wide variety of different computing devices of different configurations running different operating systems.
- [0075] Allows users to suspend and reestablish secure sessions to conserve battery power while maintaining network application sessions.
- [0076] Provides a secure solution in a wireless topology that has dead spots and coverage holes.
- [0077] No need to develop custom mobile applications or use mobile libraries to get applications to work in a mobile environment.
- [0078] Secure transport and application session persistence
- [0079] works within existing network security so the network is not compromised.
- [0080] compatible with any of a variety of conventional security protocols including for example RADIUS, Kerberos, Public Key Infrastructure (PKI), and Internet Security Protocol (IPSec).
- [0081] The computing environment and the applications do not need to change mobility is there to use but its use is transparent to the user and to the applications.
- [0082] Since all or nearly all applications run unmodified, neither re-development nor user re-training is required.
- [0083] Automatic regeneration of user-session keys at a customized interval.
- [0084] Continuous, secure connection ensuring data integrity between wired and wireless data networks.
- [0085] Enterprises running VPNs (e.g., PPTP, L2TP/IPSec, IPSec, Nortel, Cisco, other) can use these techniques to add wireless optimization, session persistence and additional security for mobile workers.
- [0086] Seamlessly integrates into enterprises where LEAP or other access point authentication security is deployed to add optimized roamable security and encryption.

BRIEF DESCRIPTION OF THE DRAWINGS

[0087] These and other features and advantages may be better and more completely understood by referring to the following detailed description of exemplary non-limiting illustrative embodiments in conjunction with drawings, of which:

[0088] FIG. 1 shows an exemplary illustrative prior art mobile IP client architecture;

[0089] FIG. 2 shows an exemplary illustrative prior art IPSec and Mobile IP architecture;

[0090] FIG. 3 shows an exemplary illustrative network protocol enveloping that may be used to meet the possible required security policies to allow network traffic to flow between a mobile node to a foreign agent through policy enforcement equipment (e.g. firewall) to the home agent and then to another communications end point;

[0091] FIG. 4 shows an example mobility architecture in accordance with a presently preferred exemplary illustrative non-limiting embodiment of the present invention;

[0092] FIGS. 5 & 5A-5F show illustrative usage scenarios;

[0093] FIG. 5G shows an exemplary client-server architecture;

[0094] FIG. 6 shows an example simplified prior art operating system security architecture;

[0095] FIG. 7 shows the example illustrative FIG. 6 architecture modified to provide secure transparent illustrative mobility functionality;

[0096] FIG. 8 shows an example illustrative run time linking sample;

[0097] FIG. 9 shows an example illustrative client policy agent hooking; and

[0098] FIG. 10 shows an example illustrative server architecture.

DETAILED DESCRIPTION OF EXEMPLARY NON-LIMITING EMBODIMENTS

[0099] FIG. 4 shows an exemplary overall illustrative non-limiting mobility architecture. The example mobility architecture includes a mobility client (MC) and a mobility server (MS). The mobility client may be, for example, any sort of computing device such as a laptop, a palm top, a Pocket PC, a cellular telephone, a desktop computer, or any of a variety of other appliances having remote connectivity capabilities. In one exemplary embodiment, mobility client MC comprises a computing-capable platform that runs the Microsoft Windows 2000/XP operating system having security (for example, IPSec) functionality but other implementations are also possible. The system shown is scalable and can accommodate any number of mobility clients and mobility servers.

[0100] In one exemplary embodiment, mobility client MC may be coupled to a network such as the Internet, a corporate LAN or WAN, an Intranet, or any other computer network. Such coupling can be wirelessly via a radio communications link such as for example a cellular telephone network or any other wireless radio or other communications link. In some embodiments, mobility client MC may be intermittently coupled to the network. The system shown is not, however, limited to wireless connectivity—wired connectivity can also be supported for example in the context of computing devices that are intermittently connected to a wired network. The wireless or other connectivity can be in the context of a local area network, a wide area network, or other network.

[0101] In the exemplary embodiment, mobility client MC communicates with the network using Internet Protocol (IP) or other suitable protocol over at least one of a plurality of possible network interfaces. In the illustrative embodiment, mobility server (MS) is also connected to the network over at least one of a plurality of possible network interfaces. The mobility server MS may communicate with one or more peers or other computing devices. The exemplary FIG. 4 architecture allows mobility client MC to securely communicate with the peer hosts via the communications link, the network and/or the mobility server MS.

[0102] In more detail, the FIG. 4 mobility server maintains the state of each mobile device and handles the session management required to maintain continuous connections to network applications. When a mobile device becomes unreachable because it suspends, moves out of coverage or changes its "point of presence" address, the mobility server maintains the connection to the network host by acknowledging receipt of data and queuing requests.

[0103] The exemplary mobility server also manages network addresses for the mobile devices. Each device running on the mobile device has a virtual address on the network and a point of presence address. A standard protocol (e.g., DHCP) or static assignment determines the virtual address. While the point of presence address of a mobile device will change when the device moves from one subnet to another (the virtual address stays constant while the connections are active).

[0104] This illustrative arrangement works with standard transport protocols such as TCP/IP—intelligence on the

mobile device and the mobility server assures that an application running on the mobile device remains in sync with its server.

[0105] The mobility server also provides centralized system management through console applications and exhaustive metrics. A system administrator can use these tools to configure and manage remote connections, troubleshoot problems, and conduct traffic studies.

[0106] The mobility server also, in the exemplary embodiment, manages the security of data that passes between it and the mobile devices on the public airways or on a wireline network. The server provides a firewall function by giving only authenticated devices access to the network. The mobility server can also certify and optionally encrypt all communications between the server and the mobile device. Tight integration with Active Directory or other directory/name service provides centralized user policy management for security.

[0107] The FIG. 4 architecture can be applied in any or all of a large and varying number of situations including but not limited to the exemplary situations shown in FIG. 5 (for brevity and clarity sake, example embodiments are described using a single network point of attachment but it will be appreciated and understood that the current invention is not to be limited to such scope and application):

[0108] At example location number one shown in FIG. 5 and see also FIG. 5A, the mobility client (depicted as a laptop computer for purposes of illustration) is shown inside a corporate or other firewall, and is shown connected to a wireless LAN (WLAN) having an access point. In this example, a private wireless network is connected to a wireline network through the mobility server. All application traffic generated on or destined for the wireless network is secured, and no other network traffic is bridged or routed to the wireless network. Using standard firewall features found in the operating system, the system can be further configured to allow only mobility traffic to be processed by the mobility server on the wireless network. In this example, the mobility client is authenticated to the mobility server. Packets flow normally between the mobility client and the mobility server, and the communication channel between the mobility client and the mobility server is protected using the conventional IPSec security protocol.

[0109] At example location number two shown in FIG. 5, the mobility client has moved into a dead-spot and lost connectivity with the network. The mobility server maintains the mobility client's network applications sessions during this time. Had the mobile client been using Mobile IP instead of the exemplary embodiment herein, the client's sessions could have been dropped because Mobile IP does not offer session persistence.

[0110] At example location number three, the mobility client has moved back into range of the corporate network on a different subnet. The mobility client acquires a new point-of-presence (POP) address on the new subnet, negotiates a new secure channel back to the mobility server using IPSec, reauthenti-

cates with the mobility server, and resumes the previously suspended network sessions without intervention from the user and without restarting the applications. This process is transparent to the mobile applications and to the application server.

[0111] At example location numbers four and five shown in FIG. 5 and see also FIGS. 5B & 5C, the mobility client has left the corporate network and roamed into range of public networks. For example, the mobile client at location 4 shown in FIG. 5 is shown in range of a conventional Wireless Wide Area Network (WWAN) wireless tower, and the mobile client at location 5 shown in FIG. 5 is shown in range of a Wi-Fi or other wireless access point "hot spot" such as found in an airport terminal, conference center, coffee house, etc. The wireless technology used for the public network need not be the same as that used inside the enterprise—since the illustrative system provides for secure roaming across heterogeneous networks. The mobility client's traffic must now pass through a corporate or other firewall. The firewall can be configured to pass IPSec traffic intended for the mobility server and/or the mobility client can be configured to use an IPSec session to the firewall. Either solution can be implemented without end-user interaction, although intervention is possible.

[0112] In the FIG. 5B example, mobility devices are connected to a diverse, public wide area network. The enterprise is also connected to the public network through a conventional firewall. The firewall is, in the exemplary embodiment, modified to allow mobility connections, specifically to the address of the mobility server. The connections are then protected by conventional security protocols such as IPSec.

[0113] In the FIG. 5C example, a private, wired network on a corporate, hospital, or other campus, and a wireless local area network supporting mobile devices connected to it through a conventional firewall. Traffic from the public to the private network that is not destined for the correct port is denied using conventional firewall rules. The firewall rules can specify either the domain ("allow access to 123.111.x:5008") or the addresses of particular mobility servers ("allow access to 123.111.22.3:1002 and 123.111.23.4:5008")—the latter approach being more secure. On a smaller campus, a single, multi-homed mobility server could be used to handle both the wired and wireless LAN traffic. Once a user is authenticated, he or she has access to the wired network. A Network Address Translator (NAT) may be used to reduce the number of public (routable) IP addresses required. In the example shown in FIG. 5C, a many-to-one relationship is provided so that mobile devices can use just one of two IP addresses instead of requiring one address each. Any traffic coming from the wireless LAN access points preferably must satisfy both the firewall rules and be cleared by the mobility server. With encryption enabled, this configuration protects the wired network while offering legitimate wireless users full, secure access to corporate data.

[0114] FIG. 5D shows an example configuration that allows users to roam securely across different networks both inside and outside of the corporate firewall. The mobility server sits behind the firewall. When the mobility client is inside the corporate firewall, connected to the wireless LAN (WLAN), and has been authenticated to the mobility server, packets flow normally and the communication channel between the mobile device and the mobility server (mobile VPN) is protected using IPSec. In this example, the Public Key Infrastructure, passwords and/or any other desired mechanism can be used to perform the key exchange for the IPSec tunnel. For added protection, WLAN access points inside the firewall can be configured to filter all protocols except for a desired one (e.g., IPSec). The mobility server acts as a VPN protecting the data as it traverses the wireless network with IPSec encryption. In this exemplary configuration, the mobility server also acts as a firewall by preventing intruders from accessing the private network. When the mobile device (client) moves into range of the corporate network on a different subnet, it acquires a new point-of-presence (POP) address on the new subnet, negotiates a new secure channel back to the mobility server using IPSec, re-authenticates with the mobility server, and resumes the previously suspended application sessions—all without user intervention being required. The applications can continue to run and the TCP or other connections can be maintained during this network transition since the network transition is transparent to the applications and the mobility server proxies communications on behalf of the mobile device during times when it is unreachable.

[0115] The FIG. 5E illustrative network configuration extends the protection of an enterprise firewall to its mobile clients. In this illustrative scenario, the mobility client is configured to use a conventional L2TP/IPSec tunnel to the firewall. IPSec filters on the mobile client can be configured to pass only authenticated IPSec packets to the mobile client's transport protocol stack and reject all other packets. The corporate firewall can be configured to reject all packets except for authenticated IPSec packets for trusted clients; any control channels necessary to set up secure connections; and responses to packets that originate from within the firewall for specifically permitted Internet or other network services. The mobility server located behind the firewall acts as a transport-level, proxy firewall. By proxying all network traffic, user transactions are forced through controlled software that protects the user's device from a wide variety of attacks including for example those using malformed packets, buffer overflows, fragmentation errors, and port scanning. Because the mobility server acts as a transport-level proxy, it can provide this protection transparently for a wide range of applications. Attacks against the network can be blocked by filter rules configured on the firewall and/or the proxy firewall capabilities of the mobility server. Attacks against the mobile device are prevented by the IPSec filter rules configured on the mobile client. Attempts to crack user passwords

using sniffer attacks are thwarted by the secure tunnel provided by IPSec.

[0116] FIG. 5F shows an additional exemplary illustrative e-commerce model. Like WAP, the FIG. 5F arrangement provides optimizations that enhance performance and reliability on slow and unreliable wireless networks. Unlike WAP, the FIG. 5F system doesn't allow data to sit on an intermediate server in an unencrypted state. The FIG. 5F architecture allows standard web protocols such as HTTP and TLS to be used for e-commerce or other transactions (the web traffic is treated as a payload). The encrypted data is forwarded to its final destination (e.g., the web server) where it can be processed in the same way it would be if two wired peers were performing the same transaction. In addition to optimizations for wireless networks, the FIG. 5F system provides seamless roaming between different networks and application session persistence while devices are suspended or out of range of a wireless base station. When combined with the illustrative system's support for public key infrastructure and/or other security mechanisms, those capabilities form a powerful mobile e-commerce platform.

[0117] The scenarios described above are only illustrative—any number of other intermittent, mobile, nomadic or other connectivity scenarios could also be provided.

[0118] Exemplary Integration With IPSec Standards-Based Security Framework

[0119] Generally, the IPSec process of protecting frames can be broadly handled by three logically distinct functions. They are:

[0120] Policy configuration and administration

[0121] Security negotiation/key management

[0122] Privacy processing

[0123] Although these processes are logically distinct, the responsibility for implementing the functionality may be shared by one or more modules or distributed in any manner within an operating or other system. For instance, in the exemplary illustrative client operating system embodiment, the implementation is broken into 3 functional areas or logical modules:

[0124] 1. A Policy Agent module

[0125] 2. A security negotiation and key management (e.g., ISAKMP/IKE) module

[0126] 3. A privacy (e.g., IPSec) module.

[0127] In this illustrative example, the Policy Agent is responsible for the configuration and storage of the configured policy—however it is the IPSec module that actually acts upon the requested policy of the Policy Agent. The preferred exemplary illustrative system provides two different related but separated aspects:

[0128] the first aspect handles IPSec from the mobility client to the firewall or the mobility server; and

[0129] the other aspect handles communication on virtual addresses between the mobility server and peer hosts.

[0130] We first discuss exemplary illustrative communication to and from the mobility client.

Example Mobility Client Architecture

[0131] As part of the preferred embodiment's overall design, network roaming activity is normally hidden from the applications running on the mobility client—and thus, the application generally does not get informed of (or even need to know about) the details concerning mobility roaming. Briefly, as described in the various copending commonly-assigned patent applications and publications referenced above, each of the mobile devices executes a mobility management software client that supplies the mobile device with the intelligence to intercept network activity and relay it (e.g., via a mobile RPC or other protocol) to mobility management server. In the preferred embodiment, the mobility management client generally works transparently with operating system features present on the mobile device to keep client-site application sessions active when contact is lost with the network. A new, mobile interceptor/redirector component is inserted at the conventional transport protocol interface of the mobile device software architecture. While mobile interceptor/redirector could operate at a different level than the transport interface, there are advantages in having the mobile interceptor/redirector operate above the transport layer itself. This mobile interceptor or redirector transparently intercepts certain calls at this interface and routes them (e.g., via RPC and Internet Mobility Protocols and the standard transport protocols) to the mobility management server over the data communications network. The mobile interceptor/redirector thus can, for example, intercept network activity and relay it to server. The interceptor/redirector works transparently with operating system features to allow application sessions to remain active when the mobile device loses contact with the network.

[0132] This arrangement provides an advantageous degree of transparency to the application, to the network and to other network sources/destinations. However, we have found that IPSec is a special case. Between the mobility client and the mobility server or the mobility client and a firewall, IPSec is protecting the packets using the point-of-presence (POP) address. Therefore, in one exemplary embodiment, to allow the existing IPSec infrastructure to operate normally, it should preferably remain informed of the current state of the network. We have therefore modified our previous design to inform IPSec of the change of network status (e.g., so it can negotiate a IPSec session when network connectivity is reestablished) while continuing to shield the networked application and the rest of the operating system from the temporary loss of a network access. Before describing how that is done in one illustrative embodiment, we first explain—for purposes of illustration only the conventional Microsoft Windows 2000/XP operating system IPSec architecture shown in FIG. 6. Note that Windows 2000/XP and IPSec is described only for purposes of illustration—other operating systems and security arrangements could be used instead.

[0133] In Windows 2000/XP, the IPSec module is responsible for filtering and protecting frames. For additional information, see for example Weber, Chris, "Using IPSec in Windows 2000 and XP" (Security Focus 12/5/01). Briefly, however, by way of non-limiting illustrative example, before allowing a frame to be processed by the protocol

stack or before transmitting the frame out on the network, the network stack first allows the IPSec module a chance to process the frame. The IPSec module applies whatever policies to the frame the Policy Agent requests for the corresponding network identity. In the event that the Policy Agent requires the IPSec module to protect a frame but it does not yet have the required security association (SA) with the peer in accordance with the requested policy, it issues a request to the security negotiation/key management module—in this illustrative case the ISAKMP/IKE (Internet Security Association and Key Management Protocol/Internet Key Exchange) module—to establish one. It is the responsibility of the ISAKMP/IKE module in this illustrative system to negotiate the requested security association and alert the IPSec (privacy) module as to the progress/status of the security association. Once the security association has been successfully established, the IPSec module continues its processing of the original frame.

[0134] In the illustrative embodiment, the Policy Agent uses conventional Microsoft Winsock API's (application programming interfaces) to monitor the state of the network and adjust its policies accordingly. However this is implementation-dependent as other interfaces may also be used to alert this logical component of the network state in other environments. Accordingly, the ISAKMP/IKE module also uses conventional Microsoft Winsock API's to perform security association negotiation as well as track network state changes in one exemplary embodiment.

[0135] Briefly, the above techniques establish a secure IPSec session that is generally tied to a particular IP address and/or port and must be essentially continuous in order to be maintained, as is well known. If the secure session is temporarily interrupted (e.g., because of a lost or suspended connection or a roam) and/or if the IP address and/or port changes, IPSec will terminate it. Unless something is done, terminating the secure IPSec session will cause the mobile application to lose communication even if the network session continues to appear to remain in place. The preferred illustrative exemplary embodiment solves this problem by introducing functionality ensuring that IPSec is passed sufficient information to allow it to react to the secure session being lost while continuing to shield this fact from the application—and by allowing IPSec to (re)negotiate a secure session once the network connectivity is reestablished using the same or different IP address or port number—all transparently to the networked application. In this way, the exemplary illustrative application is not adversely affected by termination of a previous security session and the establishment of a new one—just as the application is not adversely affected by access to the previous network being terminated and then reestablished (or in the case of roaming, to a new network with a new network identity being provisioned in its place). Meanwhile, the mobility server during such interruptions continues to proxy communications with the peer(s) the mobile device is communicating with so that network application sessions are maintained and can pick up where they left off before the interruption occurred.

[0136] Mobility client-side and server-side support each have different requirements. Therefore the architectures are different in the exemplary illustrative embodiment. The block diagram of an exemplary client architecture is shown in FIGS. 5G and 7. Note that as compared to conventional FIG. 6, we have added two additional components:

[0137] a Policy Agent Hooking component (nmplagt), and

[0138] a network virtualizing component (nmdrv).

[0139] Briefly, in the preferred illustrative embodiment, the network-virtualizing component virtualizes the underlying client module network while selectively allowing the core operating system's IPSec infrastructure to continue to be informed about network state changes. In the illustrative embodiment, the Policy Agent Hooking component "hooks" certain Policy Agent functions and redirects such processing to the network-virtualizing component so that the normal function of IPSec can be somewhat modified.

[0140] In more detail, in the exemplary embodiment, the network-virtualizing component (nmdrv) uses the services of the existing networking stack and is the layer responsible for virtualizing the underlying client module network. It also initiates and maintains the connection with the mobility server. When a client network application asks for the list of local IP addresses, the network-virtualizing component (nmdrv) intercepts the request and returns at least one of a possible plurality of the mobility client's virtual network identities (e.g. virtual IP addresses).

[0141] However, to continue to allow the inherent IPSec components to operate in a normal fashion, the client architecture should preferably allow the associated IPSec modules to see and track the current point of presence (POP) network address(es). Therefore, in the exemplary embodiment, if a request for the list of network addresses is issued and the request originated in the IPSec process, the network-virtualizing module passes the request along to an inherent network stack without any filtering or modification. Therefore, both the Policy Agent (e.g., polagent.dll in Windows 2000, ipsecsvc.dll in Windows XP) and the ISAKMP/IKE module are kept abreast of the mobility client's current POP address(es).

[0142] In the exemplary embodiment, the network-virtualizing module also tracks address changes. Without this component, the network stack would normally inform any associated applications of address list changes through the conventional application-programming interface, possibly by terminating the application communications end point. In the Microsoft operating systems, for example, this responsibility is normally funneled through the conventional Winsock module, which in turn would then inform any interested network applications of the respective changes. In the exemplary embodiment, the Policy Agent registers interest with Winsock (e.g., using the SIO_ADDRESS_LIST_CHANGE IOCTL via the conventional WSAIoctl function) and waits for the associated completion of the request. The Policy Agent may also be event driven and receive asynchronous notification of such network state changes. Again, in the illustrative exemplary embodiment, the Policy Agent also registers with Winsock a notification event for signaling (e.g., on FD_ADDRESS_LIST_CHANGE via the WSAEventSelect function). When the Policy Agent is alerted to an address list change, it retrieves the current list of addresses, adjusts its policies accordingly and updates the associated policy administration logic. It further informs the Security Negotiation/Key Exchange module, in this case the ISAKMP/IKE module, of the associated state change. The security negotiation/key exchange module (ISAKMP/IKE) module, in turn, updates its list of open connection endpoints for subsequent secure association (SA) negotiations.

[0143] In the exemplary embodiment, Winsock and associated applications are normally not allowed to see address list changes since this may disrupt normal application behavior and is handled by the network-virtualizing component. Therefore, in the preferred exemplary embodiment, another mechanism is used to inform the Policy Agent of changes with respect to the underlying network. To fulfill this requirement in the illustrative embodiment, the services of the Policy Agent Hooking module (nmplagt) are employed.

[0144] To achieve the redirection of services, the illustrative embodiment employs the facilities of a hooking module (nmplagt), and inserts the code into the policy administration, security negotiation, and key management (Policy Agent/ISAKMP/IKE) process(es) that are provided as part of the core operating system. In this illustrative embodiment, hooking only certain functions of the Policy Agent module to this redirected code is accomplished via a combination of manipulating the Import Address Table (IAT) together with the use of a technique known as code injection. Injection of the redirected functions is accomplished with the help of conventional operating system APIs (e.g. OpenProcess, VirtualAllocEx, ReadProcessMemory, WriteProcessMemory, and CreateRemoteThread) in the exemplary embodiment. In the preferred exemplary embodiment, once nmplagt.dll is injected in lsass.exe executable module, it hooks LoadLibrary and FreeLibrary entries in lsasrv.dll so it can detect when the policy agent is loaded and unloaded. Of course, other implementations are possible depending on the particular operating environment.

[0145] Furthermore, the hooking technique in the illustrative embodiment takes advantage of the way in which the Microsoft Windows itself performs dynamic run-time linking. Generally, to facilitate code reuse, Microsoft Windows supports and uses extensively, Dynamic Link Libraries (DLLs). Through the use of DLL technology, a process is able to link to code at run-time. To call a function in a dynamically linked library, the caller must know the location (address) of the specific function in the DLL. It is the operating systems responsibility to resolve the linkage between the code modules and is accomplished via an exchange of formatted tables present in both the caller and callee's run-time code modules. The dynamic library being called contains an Export Address Table (EAT). The Export Address Table contains the information necessary to find the specifically requested function(s) in the dynamic library. The module requesting the service has both an Import Lookup Table (ILT) and an Import Address Table (IAT). The Import Lookup Table contains information about which dynamic library are needed and which functions in each library are used. When the requesting module is loaded into memory for whatever reason, the core operating system scans the associated Import Lookup Table for any dynamic libraries the module depends on and loads those DLLs into memory. Once the specified modules are loaded, the requesting modules Import Address Table is updated by the operating system with the address(location) of each function that maybe accessed in each of the dynamically loaded libraries. Once again, in other environments, different implementations are possible.

[0146] In the exemplary embodiment, after the nmplagt module is loaded by the prescribed method above, it hooks the Policy Agent's calls to the conventional Microsoft

Windows Winsock functions WSASocket, WSAIocctl, WSAEventSelect, closesocket, and WSACleanup. After this process is executed, whenever the Policy Agent module attempts to register for notification of address changes, the request is redirected to the network-virtualizing component. As previously mentioned, the network-virtualizing component by design is aware of changes in network attachment. When it detects a change to the point of presence address, it sends the appropriate notifications to the Policy Agent module. In the illustrative embodiment, this causes the Policy Agent module to query for the current address list. Thus, the Policy Agent and consequently the ISAKMP/IKE module are informed of any address list changes.

[0147] FIG. 8 is an example of how in the illustrative embodiment a single function from a single DLL might be linked into a calling process. In more detail, the operating system searched the ILT, found a need for target.dll, loaded target.dll into app.exe's address space, located TargetFunc in target.dll's EAT, and fixed up app.exe's IAT entry to point to TargetFunc in target.dll. Now when app.exe calls its stubbed TargetFunc, the stub function will call through the IAT to the imported TargetFunc. Because all of the calls of interest go through the IAT, the preferred exemplary embodiment is able to hook its target functions simply by replacing the corresponding entry for each function in the IAT as shown in FIG. 9. This also has the advantage of localizing the hooking. Only the calls made by the requesting module in the target process are hooked. The rest of the system continues to function normally.

[0148] In summary, the illustrative embodiment in one exemplary detailed implementation performs the following steps:

[0149] 1. Call OpenProcess to obtain access to the policy administration, security negotiation, and key management (Policy Agent/ISAKMP/IKE) process(es) and address space

[0150] 2. Use ReadProcessMemory function to find the LoadLibrary function in the associated process(es)' address space.

[0151] 3. Using the VirtualAllocEx function, allocate enough memory to hold the inject illustrative code shown in step 4 in the specified process(es)' address space.

[0152] 4. Use WriteProcessMemory to inject the following code into the memory allocated in step 3:

[0153] LoadLibrary(&targetlibraryname);

[0154] label targetlibraryname:

[0155] "C:\\Program Files\\NetMotion client\\nmplagt.dll"

[0156] The address of the LoadLibrary function was determined in step 2. The data bytes at label targetlibraryname will vary depending on the name of the module being loaded, where the corresponding module is located, and the operating system environment.

[0157] 5. Call the CreateRemoteThread function to run the injected code.

[0158] 6. Wait for the remote thread to exit.

[0159] 7. Free the allocated memory

[0160] 8. Close the process.

[0161] At the end of these steps, the nmplagnt module has been injected into the policy administration, security negotiation, and key management (Policy Agent/ISAKMP/IKE) process(es) where it is able to redirect the processing of the needed function calls. It is understood that the above code procedure is operating system and processor dependent and is only shown for illustrative purposes, thus not limiting to this specific sequence or operation. Furthermore, the executable code responsible for adding these components to the operating environment can be provided to the mobile device via storage on a storage medium (e.g., optical disk) and/or by downloading over the network

[0162] A similar method is employed using the FreeLibrary function instead of LoadLibrary function to reverse the hooking process and to unload the nmplagnt module. For the sake of brevity, the description is kept minimal, as anyone schooled in the art should be able to achieve the desired results.

Example Mobility Server Architecture

[0163] Using IPSec methodology for communication between the mobility server and peer hosts is a different set of problems to solve—although it uses some of the same techniques used on the mobility client. FIG. 10 shows an exemplary server architecture. In the exemplary illustrative embodiment, the mobility server MS can also be based on a Windows 2000/XP (or any other) operating system. In this particular illustrative implementation, a hooking module is also used in the illustrative embodiment—but the functions intercepted by the hooking module in the case of mobility server MS are redirected to the proxy and filter modules that are also supplied by the preferred exemplary embodiment, instead of the network-virtualization module.

[0164] In the exemplary mobility server, a proxy driver (nmproxy) can be used to implement the bulk of the mobility server functionality. However, in one exemplary implementation, there are three separate problems to solve for which three additional logical modules are used. They are:

[0165] a network identity mapping driver (nmprmap),

[0166] an IPSec filter driver (nmipsec), and

[0167] the security negotiation hooking library (nmike).

[0168] The first problem is how to manage virtual addresses for the mobility clients. Although it is possible in some network stack implementations to assign multiple addresses to the inherent networking stack components of the operating system, some systems do not support such functionality. To support the more restrictive implementation, the illustrative example embodiment employs the use of an identity mapping technique. It will be appreciated that the techniques herein are both compatible with and complementary to either implementation, and such identity mapping functionality allows the security functionality to successfully operate within the more restrictive environments. The illustrative mobility server opens a communications endpoint associated with a local address and port and then identity maps between the corresponding virtual address(es)

and port(s) before packets are processed by the protocol stack during reception and before they are transmitted out on the network. That mapping is the job of the network identity mapping module (nmprmap) in one exemplary embodiment. For example, assume an application on a mobility client opens TCP port 21 on virtual address 10.1.1.2. Through the use of previously-defined mechanisms (see for example U.S. patent application Ser. No. 09/330,310 filed Jun. 11, 1999, entitled "Method And Apparatus For Providing Mobile and Other Intermittent Connectivity In A Computing Environment"), this request is transferred from the mobility client to the mobility server. In response, the MS opens the connection on its local address 10.1.1.1 on port 2042 and registers the appropriate mapping with the network identity mapping module. When the mobile client together with proxy driver (nmproxy) then wishes to send data on its newly opened connection, the packet generated by the inherent networking stack will have a source address of 10.1.1.1 port 2042. The network identity mapping module will then match the frame's protocol/address/port tuple against its mapping table and replace the source address with 10.1.1.2 port 21 before the packet is transmitted on network. The reverse operation is performed for received packets. Using this network identity mapping technique allows the mobility server to communicate to peer systems using virtualized addresses without requiring modification to the core operating system transport protocol stack

[0169] The second problem is a direct result of this mapping technique. Because the network identity mapping module logically operates below IPSec module (i.e. processes frames before during reception and after during transmission), it cannot directly manipulate IPSec protected frames without corrupting the packets or being intimately involved in the privacy or authenticating process. To address this issue, in one exemplary embodiment, the aforementioned IPSec filter module (nmipsec) inserts itself between the operating systems networking stack components and the associated IPSec modules. The filter module inspects each outgoing packet before IPSec protects the packet and each incoming packet after IPSec removes any encoding. Once in control of the frame, it consults the network identity mapping module (nmprmap) to determine whether or not the frames source or destination identity should be mapped. In this way, the functionality of the mapping logic is moved to a level where it can perform its function without interfering with the IPSec processing.

[0170] Hooking the link between the IPSec and networking stack components is implementation and operating system dependent. In the illustrative exemplary embodiment, again the hooking process is completed by the manipulation of tables that are exchanged between the inherent IPSec and networking stack modules—but other implementations and environments could rely on other techniques. In the illustrative embodiment the IPSec filter module (nmipsec) loads before the IPSec module but after the transport protocol module. When the IPSec module attempts to exchange its function table with the transport protocol components, the IPSec filter module (nmipsec) records and replaces the original function pointers with its own entry points. Once the associated tables are exchanged in this manner, the IPSec filter module (nmipsec) can manipulate the contents of and control which packets the inherent IPSec module operates on.

[0171] The third issue is where the hooking techniques also used by the mobility clients is employed. As mentioned previously, due to the mapping technique employed in one exemplary implementation, the inherent networking stack has no knowledge of the mobility client's virtual address(es). Consequently, the policy administration, security negotiation, and key management (Policy Agent/ISAKMP/IKE module) process(es) are also not cognizant of these additional known network addresses. Therefore, there are no IPSec security policies to cover frames received for or transmitted from the mobility client's virtual address(es). Furthermore the security negotiation module (in this case the ISAKMP/IKE module) has no communications end point opened for which to negotiate security associations for the mobility client. To address this issue in the exemplary embodiment, the security negotiation hooking module (nmike) can employ the same hooking methodology described for the mobility client and illustrated in FIG. 8. The security negotiation hooking module (nmike) intercepts any address change notification request. When the proxy modules registers or deregisters a mobility client's virtual address(es) with the network identity mapping module (nmprmap), it also informs the security negotiation hooking module (nmike). This module in turn then informs the policy administration module (Policy Agent) of the respective change. When either the policy administration (Policy Agent) or the security negotiation (ISAKMP/IKE) module requests a list of the current addresses via the conventional Microsoft Windows GetIpAddrTable function call, the security negotiation hooking module (nmike) intercepts the request and adds all of the current virtual addresses to the returned list. When the policy administration module (Policy Agent) sees the respective virtual addresses in the list, it treats them as actual addresses and creates the appropriate policies for the IPSec module. In response to the modification of the network address list, the security negotiation (ISAKMP/IKE) module will attempt to open and associate a communications endpoint for each address in the list. However, as mentioned previously, since the inherent networking stack in the illustrative embodiment is ignorant to the fact of these additional network addresses due to the aforementioned mapping methodology, this operation will generally fail. To solve this problem, the security negotiation hooking module (nmike) intercepts the request to the conventional Microsoft windows Winsock bind function and modifies the requested virtual address and port with a INADDR_ANY. Once the endpoint is bound through the inherent transport protocol stack, the security negotiation hooking module (nmike) employs the services of the network identity mapping module (nmprmap) and creates a mapping between the actual address and port associated with the newly established communications end point to the virtual address and the assigned port for security negotiations (in this case port 500 is the standard ISAKMP port). Finally, the security negotiation hooking module (nmike) registers the actual address and port with IPSec filtering module (nmipsec) to instruct the module to pass packets to and from the specified address without further IPSec filter processing.

[0172] All documents referenced herein are incorporated by reference as if expressly set forth herein.

[0173] While the invention has been described in connection with practical and preferred embodiments, it is not to be so limited. Specifically, for example, the invention is not limited to IPSec or Microsoft operating systems. IPSec and

related technologies can be arranged in a number of manners, executing with some of the required algorithms executing either in software or hardware. To wit, certain implementations may include hardware accelerator technology for the ciphering process, etc. Many network interface and computer manufactures have commercially available products that are used for this exact purpose. It is to be appreciated that the above specifications however describes the logical placement of required functionality and may actually execute in a distributed fashion. Accordingly, the invention is intended to cover all modifications and equivalent arrangements within the scope of the claims.

We claim:

1. A method of maintaining network communications with a mobile or other intermittently connected computing device executing at least one networked application that participates in at least one network application session, comprising:

- (a) detecting the occurrence of an event affecting network communications with the computing device, and
- (b) in response to said detection, terminating, instantiating, and/or reinstantiating an IP Security session for use by said computing device while maintaining said network application session(s).
2. The method of claim 1 wherein said detecting comprises detecting a change in network point of attachment.
3. The method of claim 1 wherein said detecting comprises detecting that an interruption of network connectivity has caused a previous IP Security session to be terminated.
4. The method of claim 1 wherein said detecting comprises detecting that the mobile device's network identity has changed.
5. The method of claim 1 wherein said detecting comprises detecting that the mobile device has roamed to a different network or subnetwork.
6. The method of claim 1 wherein said step (b) comprises negotiating a new IP Security session to replace a previous, lost IP Security session in a manner that is transparent to the networked application.
7. The method of claim 1 wherein said step (b) includes using IPSec to create a secure tunnel through the network.
8. The method of claim 1 further including applying policy rules to selectively allow, deny or delay the flow of network communications over said IP Security session.
9. The method of claim 1 further including centrally managing and distributing policy regarding the establishment of said IP Security session from a central authority.
10. The method of claim 1 further including securely proxying said mobile device communications.
11. The method of claim 1 further including terminating a previous IP Security session based on said detecting.
12. A method of modifying an operating environment having at least one software component, said operating environment using transport engine protocols and running at least one application, the method comprising:

- (a) transparently and selectively injecting computer instructions into said operating environment; and
- (b) redirecting the execution path of said at least one software component to achieve additional functionality while maintaining binary compatibility with said operating environment component(s), said transport engine protocols and said applications.

13. The method of claim 12 wherein said redirecting is performed based on process name.

14. A method of providing data communications in a mobile computing environment, said environment including at least one device using at least one network interface for network applications and operating system components, comprising:

- (a) selectively and transparently virtualizing said at least one network interface, thereby shielding said network applications and operating system components from at least some characteristics of said mobile computing environment, and
- (b) allowing other said components to remain cognizant of at least interruptions in connectivity and changes in network point of attachment.

15. A method for providing data communications in an environment including at least one device using at least one network interface for network applications and operating system components, comprising:

- (a) selectively virtualizing said at least one network interface, thereby shielding said network applications and operating system components from at least some adverse events that may otherwise disturb communications; and
- (b) using said virtualized network interface to conduct data communications.

16. The method of claim 15 wherein said adverse events include changes in network point of attachment.

17. The method of claim 15 wherein said adverse events include periods of network disconnectedness.

18. A method for using plural IP Security sessions over a plurality of network interfaces associated with at least one network point of attachment, comprising:

- (a) distributing network application communications to simultaneously flow over said plural IP Security sessions, and
- (b) multiplexing/demultiplexing said distributed communication flows into corresponding higher layer communications sessions.

19. The method of claim 18 further including applying policy rules to selectively allow, deny, or delay the flow of network communications over at least one of said plural IP Security sessions.

20. The method of claim 18 further including centrally managing and distributing policy regarding the establishment of said plural IP Security sessions from a central authority.

21. A method comprising:

- (a) facilitating the creation of plural IP Security sessions; and
- (b) selectively allowing, denying and/or delaying the flow of network communications over at least one of said plural IP Security sessions based at least in part on applying policy rules.

22. The method of claim 21 further including centrally managing and distributing said policy rules from a central authority.

23. A method of administering secure network connections comprising:

(a) establishing IP Security sessions within a computing network; and

(b) centrally managing and distributing policy regarding the establishment of said IP Security sessions from a central authority.

24. A method of proxying mobile communications comprising:

- (a) establishing communications with a mobile device;
- (b) establishing communications with an ultimate peer of said mobile device; and
- (b) instantiating at least one of a possible plurality of IP Security sessions with said ultimate peer on behalf of said mobile device.

25. The method of claim 24 wherein said mobile device includes a client and (a) comprises establishing client-server communications.

26. A method of proxying mobile communications comprising:

- (a) establishing at least one IP Security session between said mobile device and a communication peer thereof; and
- (b) maintaining said IP Security session with said communication peer during periods when said mobile device is unreachable.

27. A method of managing IP Security sessions between a mobility server and an ultimate communications peer, comprising:

- (a) establishing at least one IP Security session between said mobility server and said ultimate communications peer; and
- (b) automatically terminating said IP Security session in response to occurrence of a predetermined event.

28. The method of claim 27 wherein the predetermined event is selected from the group comprising link activity, application session inactivity, and termination of a communications end point.

29. A method of providing secure communications between a mobility client having a network identity, a mobility server and an ultimate communications peer, comprising:

- (a) establishing at least one IP Security session between the mobility server and the ultimate peer; and
- (b) securely maintaining said IP Security session even when the network identity of said mobility client changes.

30. In a system for maintaining network communications with a mobile or other intermittently connected computing device executing at least one networked application that participates in at least one network application session, said system comprising:

- a detector that detects the occurrence of an event affecting network communications with the computing device, and
- a security module that, in response to said detection, instantiates or reinstantiates an IP Security session for use by said computing device while maintaining said network application session(s).

31. The system of claim 30 wherein said detector detects a change in network point of attachment.

32. The system of claim 30 wherein said detector detects that an interruption of network connectivity has caused a previous IP Security session to be terminated.

33. The system of claim 30 wherein said detector detects that the mobile device's network identity has changed.

34. The system of claim 30 wherein said detector detects that the mobile device has roamed to a different network or subnetwork.

35. The system of claim 30 wherein said security module negotiates a new IP Security session to replace a previous, lost IP Security session in a manner that is transparent to the networked application.

36. The system of claim 30 wherein said security module uses IPSec to create a secure session through the network communication.

37. The system of claim 30 further including a policy manager that applies policy rules to selectively allow, deny or delay the flow of network communications over said IP Security session.

38. The system of claim 30 further including a central policy management authority that centrally manages and distributes policy regarding the establishment of said IP Security session.

39. The system of claim 30 further including a mobility server that securely proxies said mobile device communications.

40. The system of claim 30 wherein the security module terminates a previous IP Security session based on said detection.

41. An operating environment having at least one software component, said operating environment using transport engine protocols and running at least one application, the environment further comprising computer instructions transparently and selectively injected therein, wherein the injected computer instructions include a redirector that redirects the execution path of said at least one software component to achieve additional functionality while maintaining binary compatibility with said operating environment component(s), said transport engine protocols and said applications.

42. The environment of claim 41 wherein said redirector redirects said execution path based on process name.

43. A mobile computing environment including at least one device using at least one network interface for network applications and operating system components, said environment comprising:

(a) instructions that selectively and transparently virtualize said at least one network interface, thereby shielding said network applications and operating system components from at least some characteristics of said mobile computing environment, and

(b) further instructions that allow other said components to remain cognizant of at least interruptions in connectivity and changes in network point of attachment.

44. An environment including at least one device using at least one network interface for network applications and operating system components, said environment comprising:

instructions that selectively virtualize said at least one network interface, thereby shielding said network

applications and operating system components from at least some adverse events that may otherwise disturb communications; and

additional structure that uses said virtualized network interface to conduct data communications.

45. The environment of claim 44 wherein said adverse events include network point of attachment.

46. The environment of claim 44 wherein said adverse events include periods of network disconnectedness.

47. A system for using plural IP Security sessions over a plurality of network interfaces associated with at least one network point of attachment, comprising:

a data distributor that distributes network application communications to simultaneously flow over said plural IP Security sessions, and

(b) a multiplexer/demultiplexer that multiplexes and demultiplexes said distributed communication flows into corresponding higher layer communications sessions.

48. The system of claim 18 further including applying policy rules to selectively allow, deny, or delay the flow of network communications over at least one of said plural IP Security sessions.

49. The system of claim 47 further including a central authority that centrally manages and distributes policy regarding the establishment of said plural IP Security sessions.

50. A system comprising:

(a) a security framework that facilitates the creation of plural IP Security sessions; and

(b) a policy agent that selectively allows, denies and/or delays the flow of network communications over at least one of said plural IP Security sessions based at least in part on policy rules.

51. The system of claim 50 further including a central authority that centrally manages and distributes said policy rules.

52. A system for administering secure network connections comprising:

a security framework that establishes IP Security sessions within a computing network; and

a central authority that centrally manages and distributes policy regarding the establishment of said IP Security sessions.

53. A mobility proxy comprising:

a communications structure that establishes communications with a mobile device and with an ultimate peer of said mobile device; and

a security component that instantiates at least one of a possible plurality of IP Security sessions with said ultimate peer on behalf of said mobile device.

54. The system of claim 53 wherein said mobile device includes a client and mobility proxy comprises a server.

55. A system for proxying mobile communications comprising:

communications means for establishing at least one IP Security session with said mobile device and a communication peer thereof; and

a means for maintaining said IP Security session with said communication peer during periods when said mobile device is unreachable.

56. A system for managing IP Security sessions between a mobility server and an ultimate communications peer, comprising:

means for establishing at least one IP Security session between said mobility server and said ultimate communications peer; and

means for automatically terminating said IP Security session in response to occurrence of a predetermined event.

57. The system of claim 56 wherein the predetermined event is selected from the group comprising link activity, application session inactivity, and termination of a communications end point.

58. A system for providing secure communications between a mobility client having a network identity, a mobility server and an ultimate communications peer, comprising:

means for establishing at least one IP Security session between the mobility server and the ultimate peer and the mobility client and the mobility server; and

means for securely maintaining said IP Security session even when the network identity of said mobility client changes.

59. A storage medium storing:

a first set of instructions that inserts a policy agent hooking runtime linkable module into an operating

system having a policy agent and an IPSec infrastructure, said hooking module informing the policy agent of network state changes; and

a second set of instructions that inserts a network interface virtualizing driver into said operating system, said virtualizing driver virtualizing a client module network and initiating mobility server connections while selectively allowing the IPSec infrastructure to continue to be informed about network state changes.

60. A method of preparing a mobile device for secure communications, said mobile device having an operating environment including a policy agent and an IPSec infrastructure, said method comprising:

downloading over a computer network onto the mobile device and executing with the mobile device, a first set of instructions that insert a policy agent hooking runtime linkable module into the operating environment, said hooking module informing the policy agent of network state changes; and

downloading over the computer network and executing with the mobile device a second set of instructions that inserts a network interface virtualizing driver into said operating environment, said virtualizing driver virtualizing a client module network and initiating mobility server connections while selectively allowing the IPSec infrastructure to continue to be informed about network state changes.

* * * * *



US006941377B1

(12) **United States Patent**
Diamant et al.

(10) **Patent No.:** **US 6,941,377 B1**
(45) **Date of Patent:** **Sep. 6, 2005**

(54) **METHOD AND APPARATUS FOR
SECONDARY USE OF DEVICES WITH
ENCRYPTION**

(75) Inventors: **Nimrod Diamant, Kfar-Saba (IL);
Marcus Calescibetta, Beaverton, OR
(US)**

(73) Assignee: **Intel Corporation, Santa Clara, CA
(US)**

(*) Notice: Subject to any disclaimer, the term of this
patent is extended or adjusted under 35
U.S.C. 154(b) by 0 days.

(21) Appl. No.: **09/476,613**

(22) Filed: **Dec. 31, 1999**

(51) Int. Cl.⁷ **G06F 15/16**

(52) U.S. Cl. **709/230; 709/250**

(58) Field of Search **709/230, 105,
709/227, 250; 370/466, 463; 713/200-201**

(56) **References Cited**

U.S. PATENT DOCUMENTS

5,490,252	A *	2/1996	Macera et al.	709/249
5,963,720	A *	10/1999	Grossman	709/250
6,055,236	A *	4/2000	Nessett et al.	370/389
6,108,562	A *	8/2000	Rydbeck et al.	455/552.1
6,182,149	B1 *	1/2001	Nessett et al.	709/247
6,219,697	B1 *	4/2001	Lawande et al.	709/221
6,222,855	B1 *	4/2001	Kimber et al.	370/463
6,243,395	B1 *	6/2001	Fujimori et al.	370/395.1
6,253,321	B1 *	6/2001	Nikander et al.	713/160
6,314,525	B1 *	11/2001	Mahalingham et al.	714/4
6,321,323	B1 *	11/2001	Nugroho et al.	712/34
6,324,583	B1 *	11/2001	Stevens	709/227
6,424,621	B1 *	7/2002	Ramaswamy et al.	370/230
6,438,678	B1 *	8/2002	Cashman et al.	712/34
6,446,192	B1 *	9/2002	Narasimhan et al.	712/29
6,560,630	B1 *	5/2003	Vepa et al.	718/105
6,590,897	B1 *	7/2003	Lauffenburger et al.	370/395.6
6,708,273	B1 *	3/2004	Ober et al.	713/189
6,799,223	B1 *	9/2004	Yamamoto	709/250

2003/0074473 A1 * 4/2003 Pham et al. 709/246

OTHER PUBLICATIONS

Kent et al., RFC 2401 entitled "Security Architecture for the Internet Protocol", Nov. 1998.*

Keromytis, A.D., "Implementing IPsec" Global Telecommunications Conference, 1997. GLOBECOM '97., IEEE, vol. 3, Nov. 3-8, 1997, pp.: 1948-1952.*

Chappell, B.L., "IP security impact on system performance in a distributed real-time environment", Real-Time Systems Symposium, 1999. Proceedings. The 20th IEEE, Dec. 1-3, 1999, pp.: 218-219.*

Ogawa et al., "Smart Cluster Network (SCnet): Design of High Performance Communications for SAN", Cluster Computing, 1999, 1st IEEE Computer Society International Workshop on Dec. 2-3 1999, pp. 71-80.*

* cited by examiner

Primary Examiner—Jack B. Harvey

Assistant Examiner—Douglas Blair

(74) *Attorney, Agent, or Firm*—Steven D. Yates

(57) **ABSTRACT**

The invention provides for utilizing abilities of network interfaces, such as embedded encryption support, or access to such encryption support, so as to extend support for such abilities to network interfaces or other devices lacking such ability. In one configuration, a non-homogeneous team of network interfaces is presented to a protocol stack as being a homogeneous team, by having network interfaces lacking a particular ability be backed up by team member supporting the ability. Various methods may be applied to distribute the work load of backing up network interface according to an operation mode of the team. For example, when operating in load balancing mode, performing backup services is balanced across the team, whereas in a fault tolerant mode, processing may be first given to non-primary network interfaces.

26 Claims, 7 Drawing Sheets

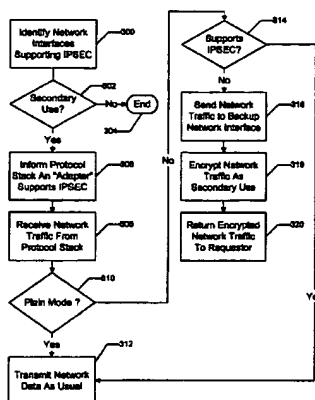


FIG. 1

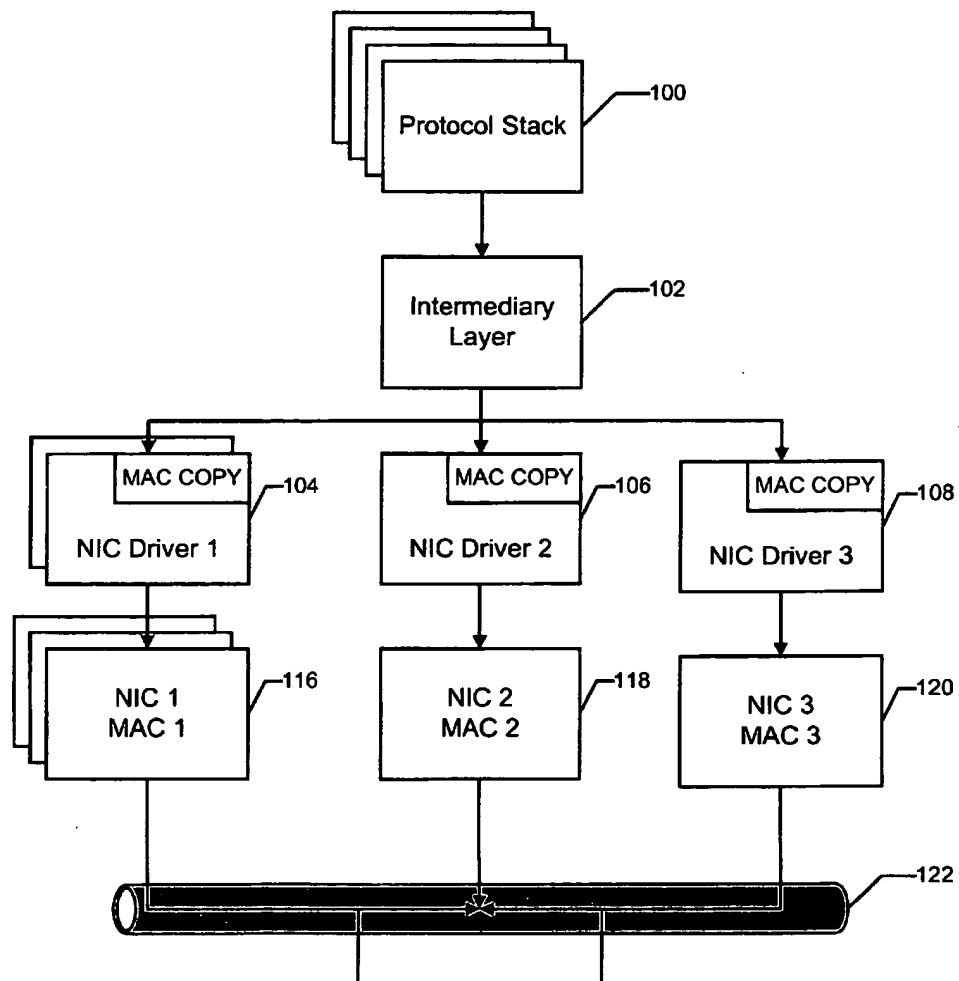


FIG. 2

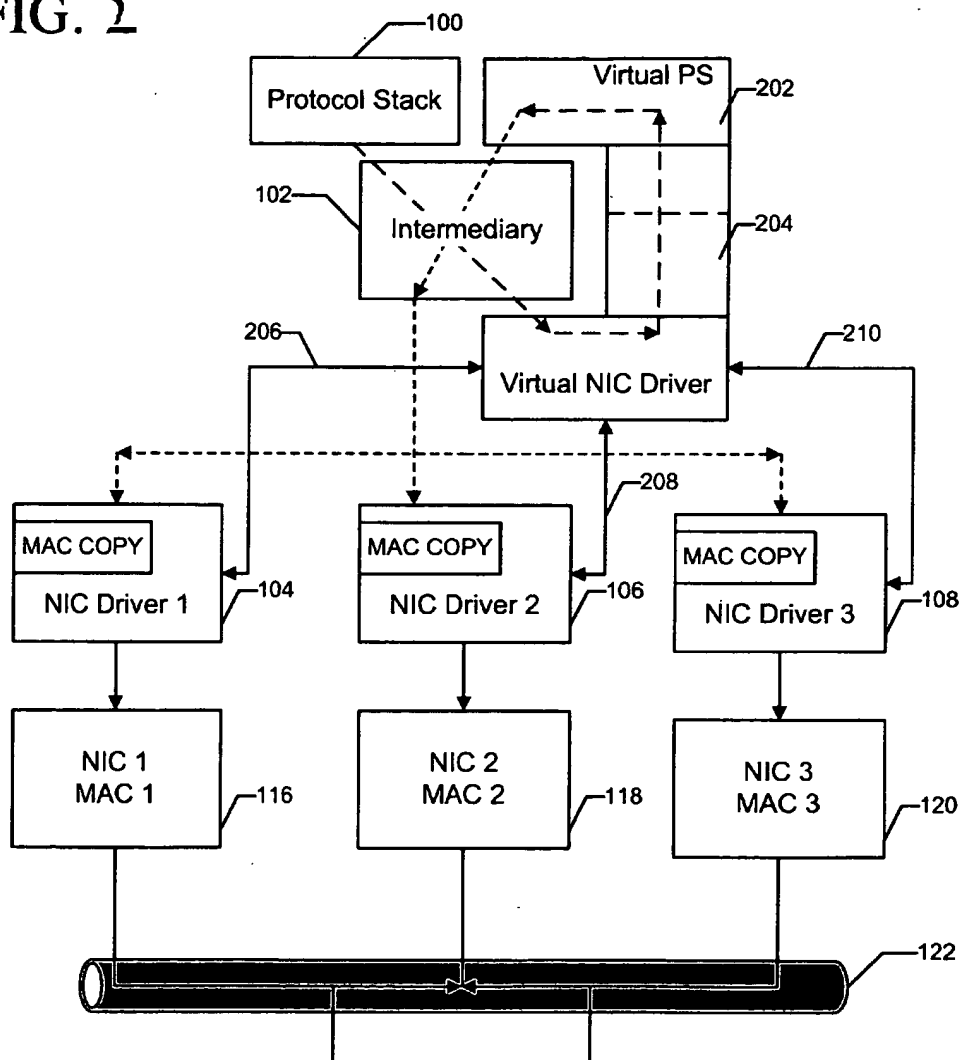


FIG. 3

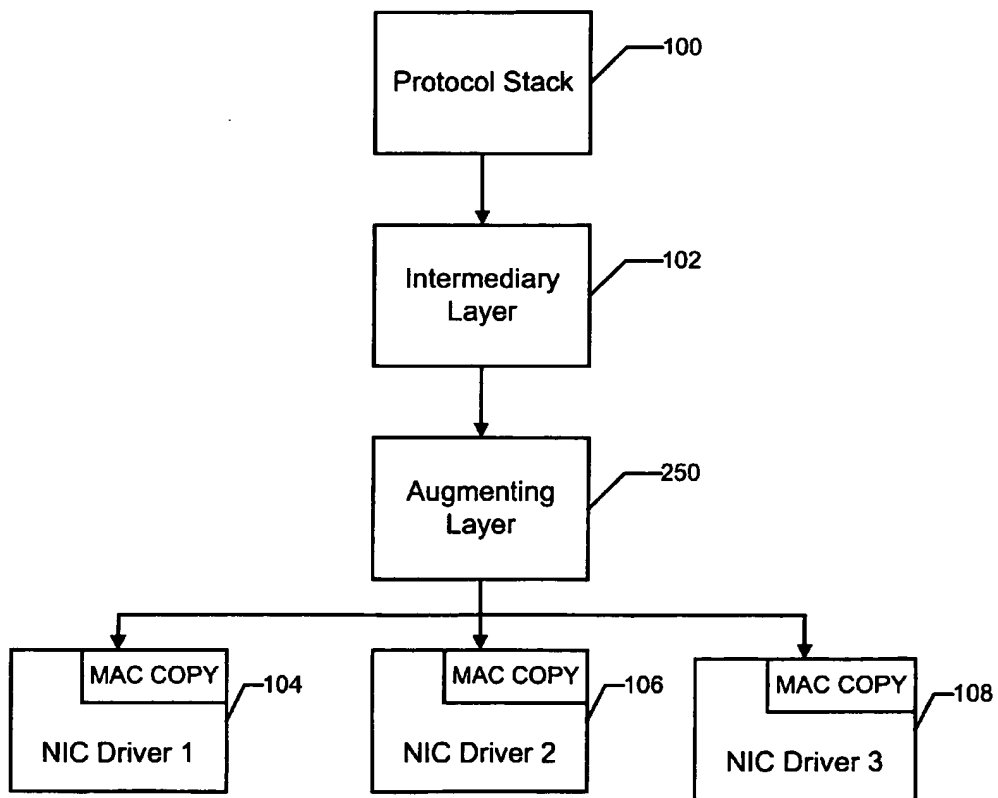


FIG. 4

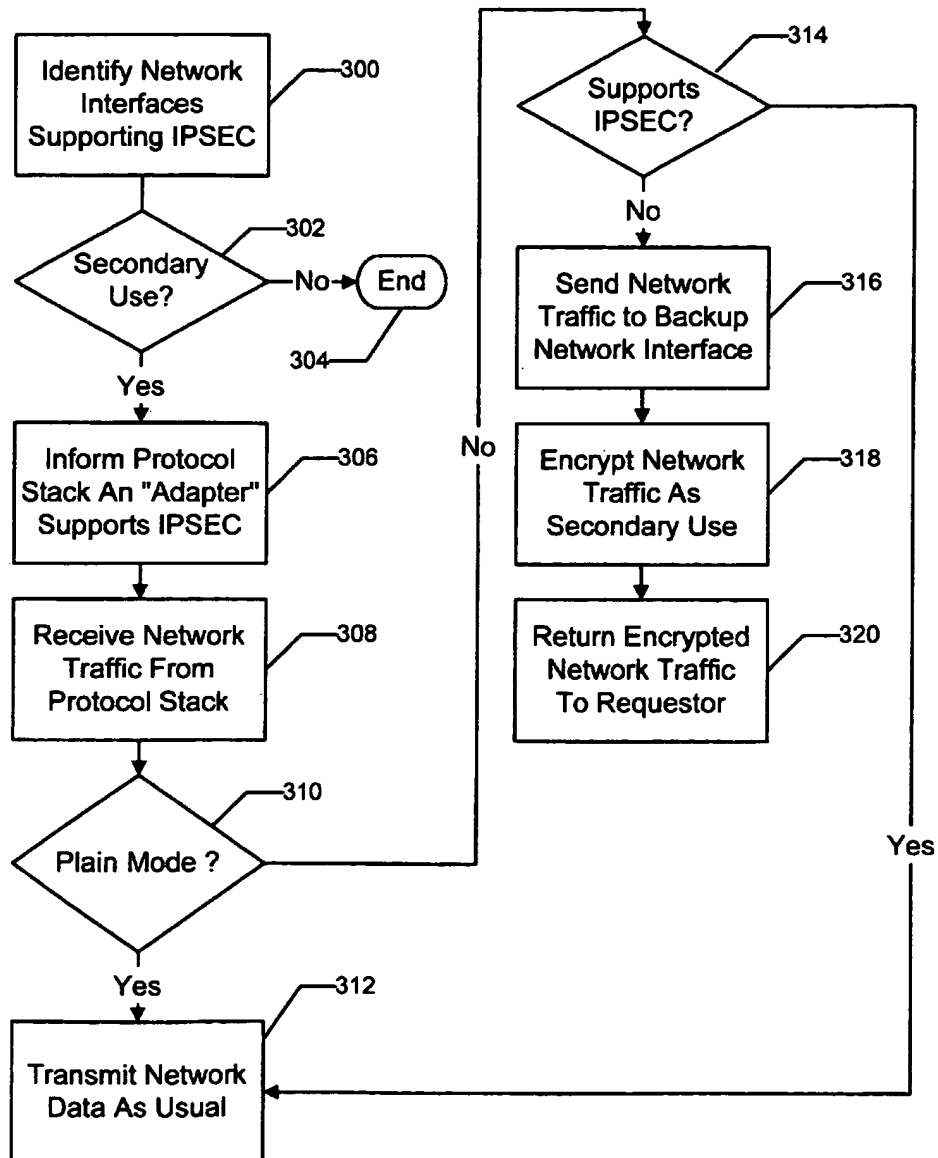


FIG. 5

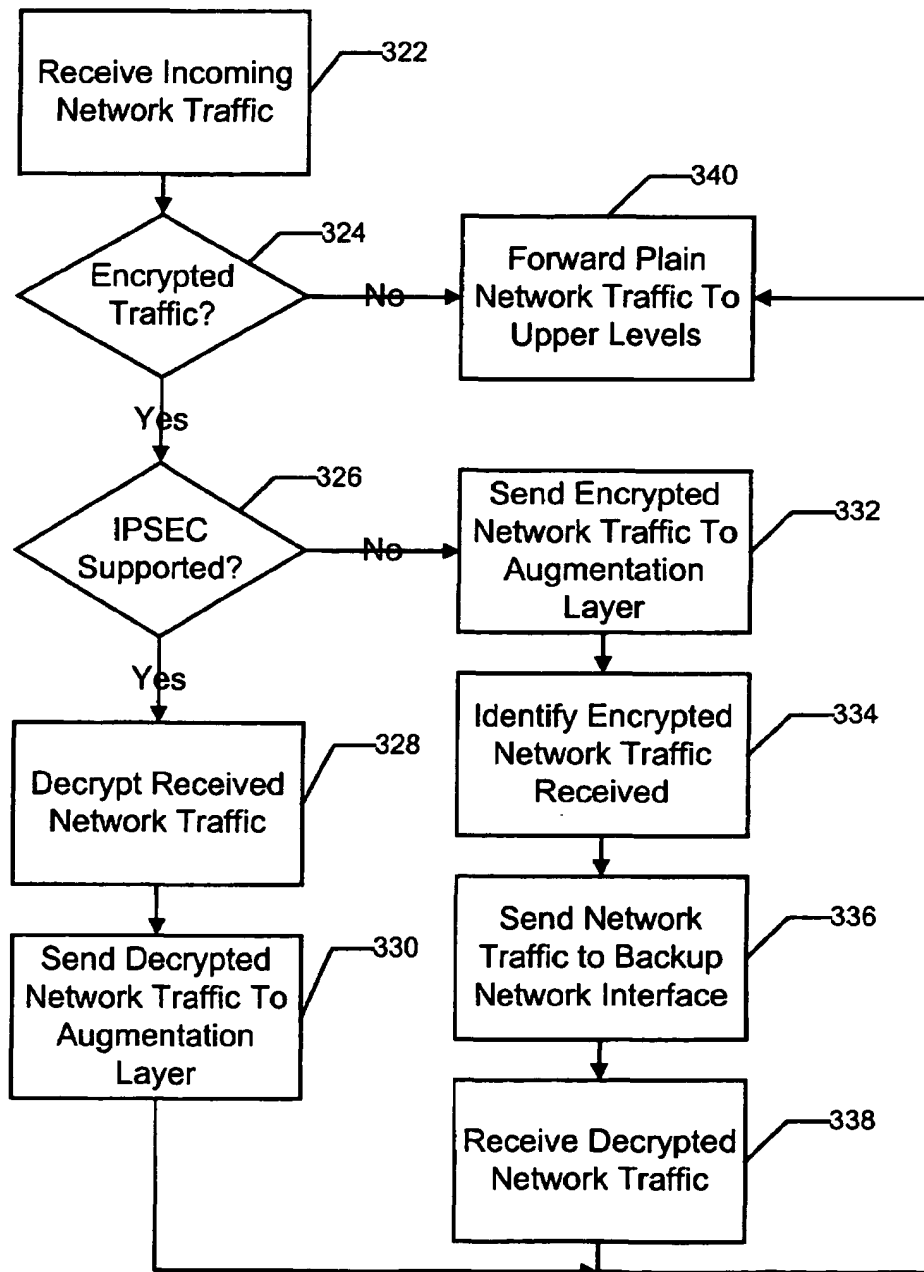


FIG. 6

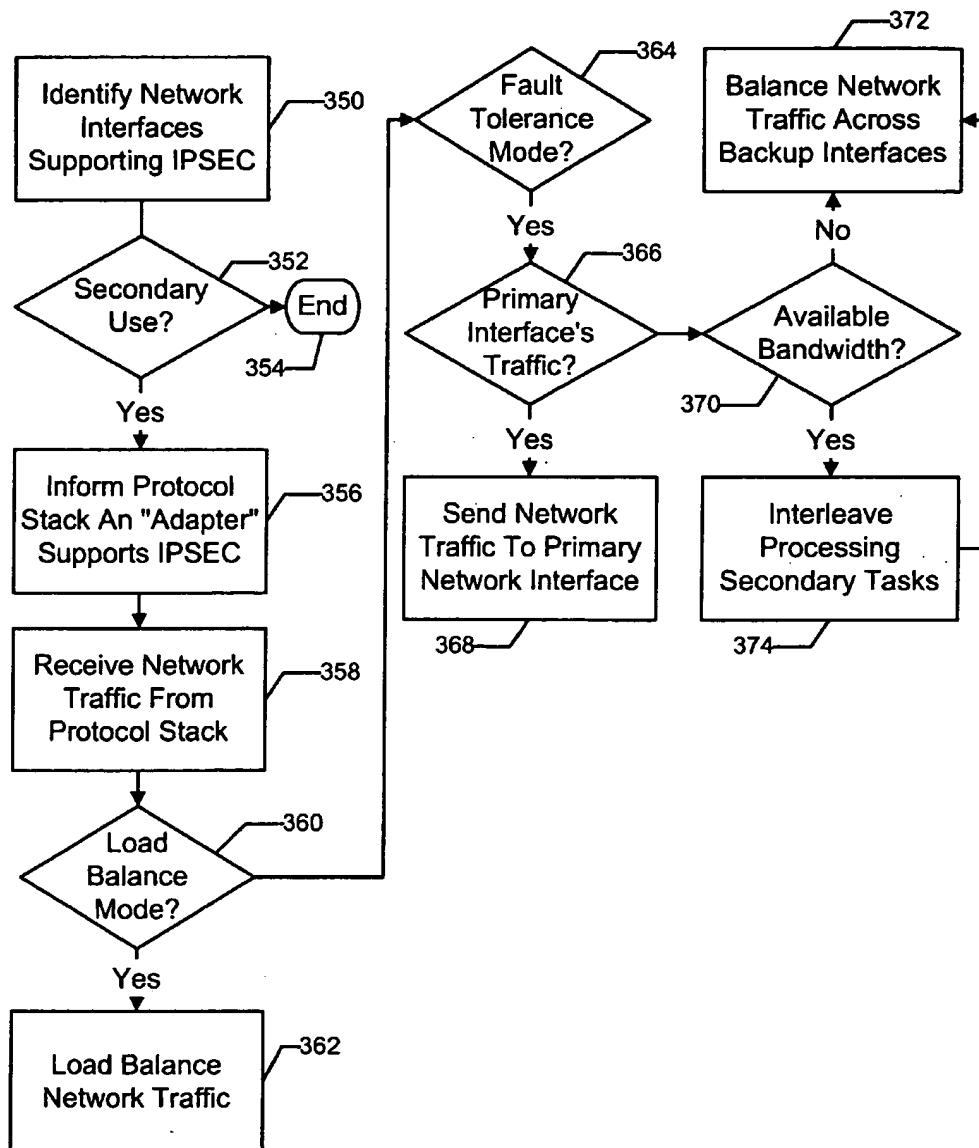
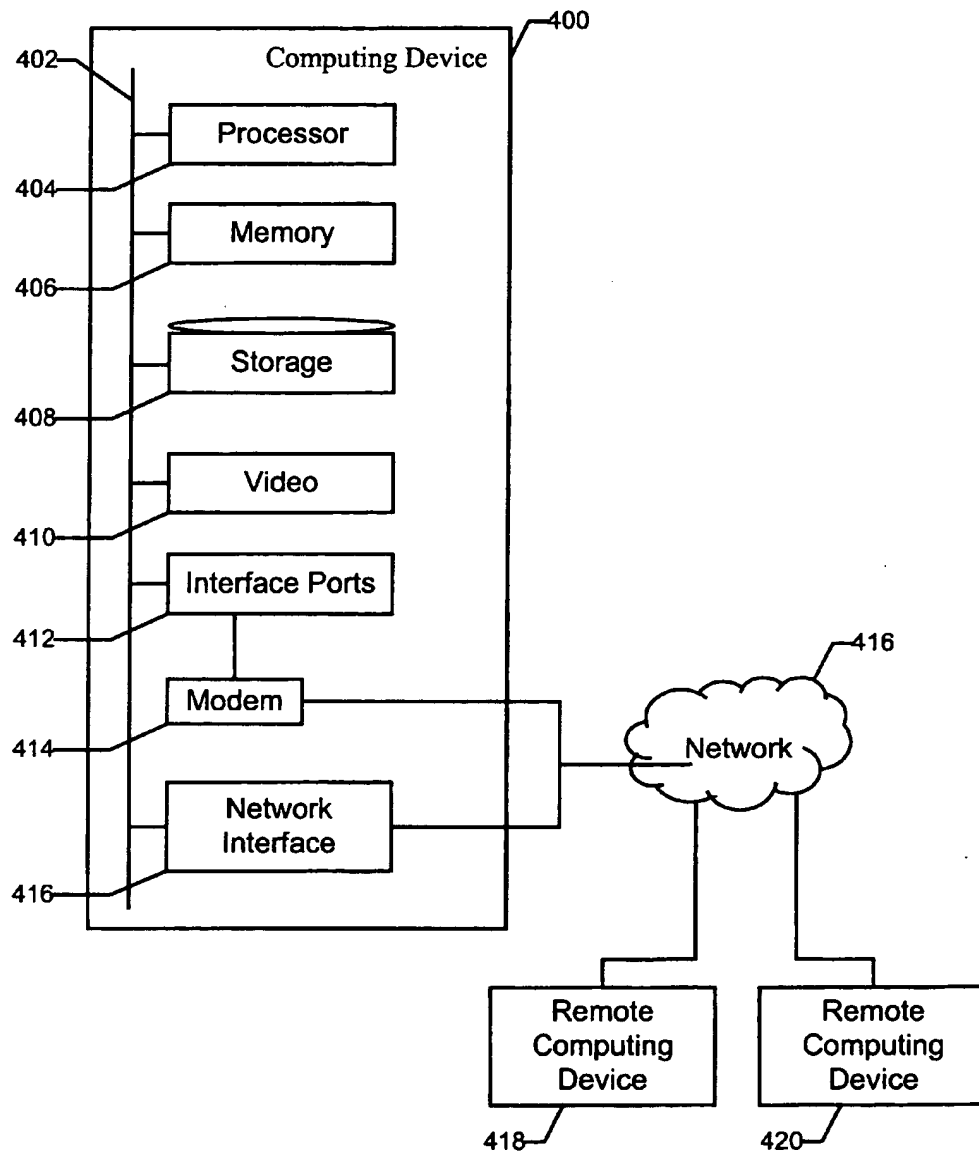


FIG. 7



1

METHOD AND APPARATUS FOR SECONDARY USE OF DEVICES WITH ENCRYPTION

FIELD OF THE INVENTION

The invention generally relates to secondary use of encryption devices, and more particularly to utilizing encryption hardware in network interface cards to provide encryption support for network interfaces lacking encryption support, and to provide parallel execution of encryption tasks by spreading such tasks across multiple network interface card encryption processors.

BACKGROUND

In conventional environments, encryption and decryption is usually performed by software. Due to the complexity involved with performing encryption, the host processor can be greatly burdened with this encryption task. This task burden is commensurate with the degree of security provided by the encryption. Unfortunately, availability of very fast computing hardware has allowed criminals to realistically apply brute-force decryption techniques to private data. Previously, typical encryption methods, such as the Data Encryption Standard (DES), used encryption key lengths of around 40–60 bits, and were considered secure.

But, as several well-publicized contests by RSA Data Security Inc. have shown, such key lengths can be compromised in a matter of days or hours. Thus, to compensate, longer key lengths (e.g., 1024 bits or higher) and more complex encryption schemes are required. This then increases the burden on the host processing system.

Such security concerns have driven efforts to provide secure networking protocols, such as Internet Protocol (IP) security, or IPSEC, promulgated by the Internet Engineering Task Force (IETF) (see IPSEC proposals at Internet location <http://www.ietf.org/ids.by.wg/ipsec.html>.) This modified IP protocol refers to encrypting IP data traffic with large key lengths and complex encryption algorithms. But, as noted above, such keys and algorithms burdens a host processor already responsible for general networking overhead, and overhead from executing other host processes.

SUMMARY

The invention provides utilization of multiple network interfaces. Network data is received for transmission by a first network interface according to a protocol. It is determined whether the first network interface supports the protocol. If the protocol is not supported, then the network data is provided to a second network interface for processing according to the protocol. The processed network data is transmitted by the first network interface.

BRIEF DESCRIPTION OF THE DRAWINGS

Features and advantages of the invention will become apparent to one skilled in the art to which the invention pertains from review of the following detailed description and claimed embodiments of the invention, in conjunction with the drawings in which:

FIG. 1 illustrates a typical network communication configuration.

FIG. 2 illustrates a low-level view of one embodiment for providing additional networking features not ordinarily supported by a network interface.

2

FIG. 3 illustrates the logical structure of a FIG. 2 embodiment.

FIG. 4 is a flowchart for using a non-homogeneous team of network adapters as a homogenous team supporting a desired protocol or functionality.

FIG. 5 is a flowchart illustrating one embodiment for processing receipt of network traffic sent according to FIG. 4.

FIG. 6 illustrates one embodiment of using a team of network interfaces to boost secondary use encryption by distributing an encryption task across multiple team members.

FIG. 7 illustrates a suitable computing environment in which certain aspects the claimed invention may be practiced.

DETAILED DESCRIPTION

In the following detailed description, numerous specific details are set forth in order to provide a thorough understanding of the present invention. However, it will be understood by those skilled in the art that the present invention may be practiced without these specific details. In other instances well known methods, procedures, components, and circuits have not been described in detail so as not to obscure the present invention.

The increasing burden of performing secured encryption with long keys and complex algorithms provides an opportunity for developers to provide a way to offload encryption burdens from a host's processor. In one embodiment, network interface developers couple an encryption processor with their network interfaces that can be used to encrypt/decrypt network traffic, as well as to provide encryption services to external hardware and processes. In one embodiment, a driver for network interfaces provides access to the encryption hardware, so as to allow external hardware and processes to avoid replying on software encryption methods. Note that the encryption processor may be physically packaged with a network interface, e.g., by way of an encryption application specific integrated circuit (ASIC) (or equivalent) on a network interface, or packaged separately and communicatively thereto.

FIG. 1 illustrates a typical network communication configuration, in which a protocol stack 100 is in communication with an intermediary layer 102 (e.g., LSL or NDIS). There may, as illustrated, be several protocol stacks 100. It is assumed there is only a single protocol stack and a single intermediary layer. The protocol stack corresponds to typical networking protocols such as TCP, IP, SPX, IPX, NetBios, Netbeui, AppleTalk, X.400, and the like. The intermediary layer 102 is bound to the protocol stack, and helps route network traffic.

The intermediary layer is in communication with multiple network interface card base drivers 104–108. As shown, instances of a single base driver 104 can be managing multiple network interfaces (three such interfaces are illustrated as a stack of interfaces 116). For presentation clarity, it is assumed each base driver communicates with a single network interface. Note that although network interface cards, or “NICs”, are shown, the term NIC is meant to include other input/output interfaces for alternate network configurations, such networks effected over serial/parallel port connections, Universal Serial Bus (USB) links, IEEE 1394 FireWire link, and the like.

In the illustrated configuration, the intermediary 102 appears to the stack 100 as a multiplexer to the different base drivers. The stack and base drivers are bound to the inter-

3

mediary, resulting in network data received by the protocol stack being routed to the intermediary. The intermediary becomes responsible for forwarding the network data on to an appropriate base driver 104-108 which is then responsible for transfer of the data to the NIC hardware 116-120 for delivery over a network connection 122.

On data reception over the network 122, all NICs see the data, but only the NIC hardware with the appropriate matching MAC filter responds to the incoming data. If a NIC accepts network data, it is forwarded to its driver, which in turn forwards it to the intermediary layer which multiplexes the data to an appropriate protocol stack.

The intermediary layer is capable of accepting many upper-layer protocol stacks, in addition to multiple drivers below it. Although not provided by present networking environments, this ability of the intermediary layer provides an opportunity for allowing transparent fail-over, load-balancing, and support for new network protocols and features, without changing existing base drivers 104-108 for current network interfaces 116-120.

FIG. 2 illustrates a low-level view of one embodiment for providing additional networking features not ordinarily supported by a network interface. FIG. 3 illustrates the logical structure of the FIG. 2 embodiment. In effect, FIG. 2 provides an "augmenting layer" 250 between a traditional intermediary layer 102 and its network interface drivers 104, 106, 108, providing opportunity to augment network interface drivers with functionality not originally planned for network interfaces 116, 118, 120.

In one embodiment, an augmentation layer 250 is implemented by "surrounding" an Intermediary layer 102 with a virtual protocol stack 202 and a virtual NIC driver 204. However, it will be appreciated by those skilled in the art that other configurations may be used to achieve a similar augmentation layer effect. (Note that this figure is highly abstracted to show general structure, and not implementation details.) A protocol stack 100, such as one typically provided by an operating system vendor (or by a network interface vendor supporting the network interface), is bound to the intermediary layer 102 in a conventional manner. The intermediary layer 102 is bound to the virtual NIC driver 204 instead of drivers 104, 106, 108 as depicted in FIG. 1. From the perspective of protocol stack 100, the protocol stack is bound to a valid network interface.

The virtual driver 204 routes networking requests to the virtual protocol stack 202 which then repackages the network traffic for the different NIC drivers 104, 106, 108. It will be appreciated that in accord with typical networking practices, return data will follow an inverse data path, allowing decryption of encrypted return data before the decrypted payload is given to the protocol stack 100. However, before routing the networking traffic to NIC drivers 104, 106, 108, the virtual driver 204, the driver may make use of original driver capabilities (e.g., ability to ask a network interface to encrypt data) by way of communication links 206, 208, 210.

Assume, for example, that NIC 1 116 has an on-board encryption ASIC, but NIC 2 118 and NIC 3 120 do not. As will be discussed in more detail below, in such a circumstance, encryption for NIC 2 118 and NIC 3 120 can be supported by routing encryption requests through NIC 1 116 encryption hardware and then repackaging the resultant encrypted data for delivery to NIC 2 118 and/or NIC 3 120 by way of the virtual protocol stack 202. That is, in one embodiment, network traffic to be encrypted would go from protocol stack 100, to the intermediary 102, to the virtual driver 204, which communicates with the NIC 1 driver 104

4

to have NIC 1 116 perform the encryption. The encrypted data is received by the virtual driver 204, given to the virtual protocol stack 202, which then re-sends the data for transmission by NIC 2 118 or NIC 3 120.

FIG. 4, is a flowchart illustrating using network interfaces to provide missing features, e.g., encryption, for other network interfaces, so as to provide a team of network interfaces apparently capable of homogeneously performing a function even though some of the network interfaces in fact cannot perform the function.

Assume the team is performing adapter fault tolerance (AFT) or adaptive load balancing (ALB), such as provided by the Intel Advanced Networking Services (iANS), and that the team is to be presented as capable of homogeneously providing IPSEC encryption support even though one or more members of the team does not have encryption support.

The phrase "Adapter Fault Tolerance" means presenting, to protocol stacks, several network interfaces (working as a team) as one network interface. One of these network interfaces acts as an active, or primary, network interface for network communication. When a fault in one of the underlying network interfaces of the team is detected, iANS switches the faulty member network interface with another member network interface known to be functional. Using AFT, network communication will be resilient to failure in the member network interface in use when fail-over to another functional member network interface occurs.

The phrase "Adaptive Load Balancing" means presenting, to protocol stacks, several network interfaces (working as a team) as one network interface, using all of the network interfaces as an active network interface for network communication. Outband network traffic (transmit) is balanced between all team members comprising a fat channel capable to deliver high bandwidth. When a fault in one of the underlying network interfaces of the team is detected, iANS does not use the adapter, providing opportunity to replace the interface.

Note that IPSEC, AFT, and ALB are presented for exemplary purposes only, and that other encryption standards and networking capabilities are also intended to be supported as discussed herein.

In one embodiment, at least one of the network interfaces is based on an Intel 82559 or similar chipset providing IPSEC encryption support for a primary and a secondary use of the adapter. Primary use corresponds to use of a network interface to transmit and receive its own network traffic. Secondary use corresponds to use of a network interface to process data for an external entity, e.g., driver software for a different network interface, operating system component, API, or the like.

In secondary use, a network interface receives data from a requestor to be encrypted or decrypted. In one embodiment, the received data is processed and returned to the requestor. In another embodiment, the processing adapter processes and then directly transmits the data to the network for the requestor. For example, timing, throughput, or other considerations, may make direct transmission more efficient than returning the data for subsequent transmission. In one embodiment, the processing adapter is instructed to temporarily change its MAC address to the MAC address of the requestor's network interface lacking encryption support, so that responses to the transmitted network data will be received by the requestor's networking interface. Accordingly, network interfaces without IPSEC support may nonetheless process IPSEC network traffic by having the encryption processing handled by an IPSEC capable device.

5

The data to be secondarily processed can be stored in a host memory, such as in a main memory for a computing device housing the network interface, copied to a memory of the network interface, or stored in some other memory and made available to the network interface. It is assumed that Direct Memory Access, private or public bus, or some other communication pathway is available for receiving and returning data. Secondary use is intended to replace software encryption functions and consequently offload work from a host processor. When network interfaces having encryption support are present within a computing device, software encryption libraries can forward encryption tasks to the interfaces to be secondarily processed by the encryption hardware, interleaved with regular network traffic that goes out to the network.

Thus, to augment adaptive load balancing, adapter fault tolerance, or other networking functionality, a first operation is to identify 300 network interfaces bound to the augmentation layer 250 support IPSEC (or other functionality) to be shared. In one embodiment, the identification 300 operation confirms network interface identity data, such as vendor information and/or revision version of the network interface, to ensure compatibility with the augmentation layer. In a further embodiment, the augmentation layer refuses to operate with network interfaces not having particular identity data. For example, in such configurations, the augmentation layer may choose to only operate with network interfaces provided by the developer of the augmentation layer software and/or drivers.

A second operation is to verify 302 that at least one IPSEC capable interfaces provides secondary-use access to its encryption hardware. A single, fast, encryption component to an adapter may support encryption requirements for many other hardware devices. Alternatively, as discussed for FIG. 5, if multiple encryption-capable adapters are present, then all adapters can share task processing, e.g., operating as parallel processors.

If verification fails, then an adapter team cannot be heterogeneously shared, and sharing terminates 304. If verification succeeds, then the augmentation layer presents 306 itself to a protocol stack (e.g., protocol stack 100) as a network interface supporting IPSEC (or other desired functionality) with support for secondary use of its encryption hardware. Additionally, the augmentation layer may announce itself to an operating system as supporting secondary-use encryption tasks, thus allowing operating system APIs (e.g., Microsoft Windows CryptoAPI) to utilize encryption capabilities of the network interfaces.

The protocol stack then delivers 308 packets for transmission to the network 122 in either plain mode or encrypt mode. If 310 plain packets are to be sent, then the packets can be presented to an appropriate network interface's driver for transmission 312 in a customary manner. (Or they can be routed through the augmentation layer without any augmentation.)

However, if the packets are to be encrypted, then for each adapter that is to receive data for transmission, a check 314 is made to determine whether the adapter supports IPSEC transmissions. Note that depending on how one tracks which adapters can perform IPSEC transmissions, this check may or may not be literally performed. For example, a transmission mask may be employed to control which adapters simply send traffic without further review. It will be appreciated that which adapters receive data depends on transmission mode; thus, for example, under load balancing, all adapters receive a distributed portion of network traffic for transmission.

6

If the destination adapter does not support IPSEC, then the data payload for the destination adapter is sent 316 to a backup adapter that does support IPSEC. The backup adapter receives the data payload, encrypts 318 it pursuant to IPSEC, and returns 320 the encrypted data for delivery by the destination adapter as regular data. This arrangement allows load balancing (or other teaming algorithms) of IPSEC or other network traffic across a non-heterogeneous adapter team.

FIG. 5 is a flowchart illustrating one embodiment for processing receipt of network traffic sent according to FIG. 4. Generally, on receipt 322 of incoming network traffic, an inverse to FIG. 4 series of operations is performed. For example, assuming a networking mode of transmitting load balanced IPSEC traffic, if 324 an encrypted packet is received from a network, and if 326 received by a network interface which is IPSEC capable, then the received traffic will automatically be decrypted 328 by the adapter and presented 330 to the augmentation layer as a plain text packet. However, if the adapter is not IPSEC capable, then encrypted packets received by the adapter will be presented 332 to the augmentation layer still in encrypted form as received from the network.

The augmentation layer identifies 334 the encrypted packets as being encrypted, and forwards 336 them for decryption (e.g., as a secondary task) by an available IPSEC-capable adapter. Decrypted packets are received 338 and forwarded 340 by the augmentation layer in accord with a current processing algorithm, e.g., traditional (direct), fault tolerant, load balancing, etc., for presentation as regular plain text packets for processing by upper layer protocol stacks.

FIG. 6 illustrates an algorithm for using a team of network interfaces, controlled by an augmentation layer 250, to boost secondary use encryption by distributing an encryption task across multiple team members; in one embodiment, the proportional distribution of the task is according to a current workload of each network interface of the team.

Secondary use encryption throughput is therefore scaled according to the number of members in the team and their availability. Secondary use in adaptive load balancing mode can be performed by distributing encryption tasks to team members according to their current workload. Secondary use in adapter fault tolerance mode favors distributing encryption tasks to network interface team members which are inactive and waiting on failure of a primary running network interface. Such idle network interfaces can be used as dedicated encryption devices.

Note that spreading processing of encryption using load balancing techniques is not limited only to using network interfaces as hardware accelerators, but also to using other hardware devices which are capable of performing encryption, such as other encryption-capable devices within a computing device hosting the network interfaces. Additionally, note that load balancing and fault tolerance are used as exemplary operations that respectively utilize all network interfaces, or a single interface of a team. It is contemplated that the present invention will be applied to other tasks.

Operations 350-358 correspond to operations 300-308 of FIG. 4, and are only briefly discussed for FIG. 5. Thus, a first operation is to identify 350 network interfaces bound to the augmentation layer 250 support IPSEC (or other functionality) to be shared, and a second operation is to verify 352 that multiple IPSEC capable interfaces provides secondary-use access to its encryption hardware. If verification fails, then encryption processing cannot be spread across the identified adapters, and spreading terminates 354. If veri-

cation succeeds, then the augmentation layer presents 356 itself to a protocol stack as a network interface supporting IPSEC with support for secondary use of its encryption hardware, and may announce itself to an operating system.

The protocol stack then delivers 358 packets for transmission to the network 122 in either plain mode, encrypt mode, or for secondary processing with loop back to a requestor (e.g., a protocol stack, encryption library or service, operating system, etc.). If 360 a network interface team is operating in adaptive load balance mode, the augmentation layer load balances 362 network traffic according to the network interface team's mode of operation.

If 364 a network interface team is operating in adapter fault tolerance mode, and if 366 regular network traffic, plain or encrypted is to be delivered to the primary (e.g., active) network interface, then the packets are delivered 368 to the primary adapter and transmitted to the network in a customary fashion.

If, however, non-regular traffic is received, e.g., secondary use data packets, then these packets are delivered to the backup network interface members such that they are balanced 372 across all available unused team members. If 370 the primary network interface has available resources, however, to process encryption tasks, then the primary adapter interleaves 374 secondary task processing with its primary transmission and receipt of network traffic. Remaining task processing is balanced 372 across all available unused team members. It is expected that appropriate queuing strategies will be employed to keep all adapters busy.

On receipt of network traffic, if the network interface team is operating in adaptive load balancing mode, or some other mode utilizing all network interfaces in the team, then if regular network traffic (plain or encrypted) is received, then it will be balanced across all team members as normal. If non-regular traffic is received, e.g., secondary use data packets, these packets are delivered to the all members of the network interface team such that they are balanced across all available team members.

Note that since encryption duties are separate from network transmission and reception, even if a network interface is defective or otherwise unable to process network transmissions, the network interface may still be functionally available for processing secondary use data. In one embodiment, when there are network interfaces that are not processing (or can not process) regular network traffic, these adapters will be first loaded with secondary use tasks to leave fully functional network interfaces available for processing regular network traffic. In addition, although not shown in these figures, processing accounts for the hot-swap removal and replacement of network interfaces. For example, if a defective network interface is replaced with a fully functional one, then the replacement interface should no longer receive a disproportionate amount of secondary use processing requests.

FIG. 7 and the following discussion are intended to provide a brief, general description of a suitable computing environment in which portions of the invention may be implemented. An exemplary system for implementing the invention includes a computing device 400 having system bus 402 for coupling together various components within the computing device. The system bus may be any of several types of bus structures, such as PCI, AGP, VESA, etc. Typically, attached to the bus 402 are processors 404 such as Intel Pentium® processors, programmable gate arrays, etc., a memory 406 (e.g., RAM, ROM, NVRAM), computing-device readable storage-media 408, a video interface 410, input/output interface ports 412, and a network interface. A

modem 414 may provide an input and/or output data pathway, such as for user input/output, and may operate as a network interface in lieu of or in conjunction with other network interfaces 416.

The computing-device readable storage-media 408 includes all computing device readable media, and can provide storage of programs, data, and other instructions for the computing device 400 and components communicatively coupled thereto (e.g., a network interface card attached to the system bus 402). Media 408 includes hard-drives, floppy-disks, optical storage, magnetic cassettes, tapes, flash memory cards, memory sticks, digital video disks, and the like.

The exemplary computing device 400 can store and execute a number of program modules within the memory 406, and computing-device readable storage-media 408. The executable instructions may be presented in terms of algorithms and/or symbolic representations of operations on data bits within a computer memory, as such representation is commonly used by those skilled in data processing arts to most effectively convey the substance of their work to others skilled in the art. Here, and generally, an algorithm is conceived to be a self-consistent sequence of steps leading to a desired result. The steps are those requiring physical manipulations of physical quantities, and can take the form of electrical or magnetic signals capable of being stored, transferred, combined, compared, and otherwise manipulated. Appropriate physical quantities of these signals are commonly referred to as bits, values, elements, symbols, characters, terms, numbers, or the like.

The invention may therefore be described by reference to different high-level program constructs and/or low-level hardware contexts, and may be part of single or multiprocessing host computing devices, such as personal computers, workstations, servers, etc., as well as hand-held devices and controllable consumer devices such as Personal Digital Assistants (PDAs), cellular telephones, or Internet television adapters. It will be appreciated that the invention can have its own processors, such as the Intel 82559 chipset providing IPSEC encryption support for network interfaces, and that these processors may operate asynchronously to, and possibly in conjunction with, host processors.

The computing device 400 is expected to operate in a networked environment 416 using logical connections to one or more remote computing devices 418, 420. In addition, the invention itself may operate in a distributed fashion across a network, where input and output, including user input and output (e.g., a graphical interface) may each occur at different networked locations. Thus, for example, assuming a perspective where computing device 400 utilizes a team of load balancing network interfaces, then remote computing devices 418, 420 include routers, a peer devices, a web server or other program utilizing networking protocols such as TCP/IP, IPSEC, IPX, hypertext transport protocol (HTTP), File Transfer Protocol (FTP), Gopher, Wide Area Information Server (WAIS), or the like.

It is understood that remote computing devices 418, 420 can be configured like computing device 400, and therefore may include many or all of the elements discussed for computing device 400. It should also be appreciated that computing devices 400, 418, 420 may be embodied as a single devices, or as a combination of separate devices; for example, a team of network interfaces may reside in a separate enclosure and be communicatively coupled to computing device 400 (e.g., by input/output interface ports 412 or other communication medium).

Having described and illustrated the principles of the invention with reference to illustrated embodiments, it will be recognized that the illustrated embodiments can be modified in arrangement and detail without departing from such principles. For example, while the foregoing description 5 focused, for expository convenience, on using encryption hardware present in network interfaces to emulate encryption in non-capable network interfaces, and on distributing encryption tasks among multiple network interfaces, it will be recognized that the same techniques and analyses discussed above can be applied to other protocols and services. In particular, the encryption support need not reside in network interfaces, and instead may be provided by other components within a computing device.

And, even though the foregoing discussion has focused on particular embodiments, it is understood that other configurations are contemplated. In particular, even though the expressions "in one embodiment" or "in another embodiment" are used herein, these phrases are meant to generally reference embodiment possibilities, and are not intended to limit the invention to those particular embodiment configurations. These terms may reference the same or different embodiments, and unless indicated otherwise, are combinable into aggregate embodiments. Consequently, in view of the wide variety of permutations to the above-described 10 embodiments, the detailed description is intended to be illustrative only, and should not be taken as limiting the scope of the invention. Rather, what is claimed as the invention, is all such modifications as may come within the scope and spirit of the following claims and equivalents thereto.

What is claimed is:

1. A method for sharing processing capabilities of utilizing multiple network interfaces among said network interfaces, comprising:

receiving a first network data to be transmitted by a first network interface according to a protocol;
determining the first network interface lacks hardware supporting the protocol;
providing said first network data to a second network interface different from the first network interface, the second network interface including hardware supporting the protocol;
transparently processing of said first network data by the second network interface into a second network data 40 according to the protocol; and
transmitting said second network data with said first network interface.

2. The method of claim 1, wherein the first network interface does not support the protocol, the method further 50 comprising:

presenting said first and second network interfaces to a protocol stack as being a homogeneous team of network interfaces.

3. The method of claim 1, wherein the protocol includes encrypting the first network data before submitting said first network data to a network.

4. The method of claim 1, further comprising:
communicatively coupling a hardware-based encryption processor with said second network interface, said encryption processor performing said processing of said first network data.

5. The method of claim 4, wherein the hardware-based encryption processor supports a primary mode for encrypting network data for said second network interface, and a secondary mode for encrypting network data for said first network interface. 65

6. The method of claim 5, wherein the said first and second network interfaces operate in an adaptive load balancing mode, and wherein said second network interface interleaves said primary mode encryption with said secondary mode encryption.

7. The method of claim 6, further comprising:
providing a third network interface supporting the protocol;

wherein processing said first network data into said second network data is balanced across said second and third network interfaces.

8. The method of claim 7, wherein said balancing is performed according to a workload of said second and third network interfaces.

9. The method of claim 5, wherein the said first and second network interfaces operate in an adapter fault tolerance mode, and wherein said first network interface is a primary network interface, and said second network interface is a backup network interface.

10. The method of claim 1, wherein the said first and second network interfaces operate in an adaptive load balancing mode, and wherein said second network interface interleaves processing network data for said second network interface with processing said first network data into said second network data.

11. The method of claim 1, wherein the said first and second network interfaces operate in an adapter fault tolerance mode, and wherein said first network interface is a primary network interface, and said second network interface is a backup network interface.

12. A readable medium having encoded thereon instructions for sharing processing capabilities of multiple network interfaces among said network interfaces, the instructions capable of directing a processor to:

receive a first network data to be transmitted by a first network interface according to a protocol;
determine the first network interface lacks hardware supporting the protocol;
provide said first network data to a second network interface different from the first network interface, the second network interface including hardware supporting the protocol;
transparently process said first network data by the second network interface into a second network data according to the protocol; and
transmit said second network data with said first second network interface.

13. The medium of claim 12, wherein the protocol includes encrypting the first network data before submitting said first network data to a network.

14. The medium of claim 12, said instructions including further instructions to direct said processor to:

process said first network data into said second network data with a hardware-based encryption processor communicatively coupled with said second network interface.

15. The medium of claim 14, wherein the hardware-based encryption processor supports a primary mode and a secondary mode, said instructions including further instructions to direct said processor to:

encrypt network data for said second network interface when said encryption processor is in said primary mode; and
encrypt network data for said first network interface when said encryption processor is in said secondary mode.

16. The medium of claim 15, wherein said first and second network interfaces operate in an adaptive load balancing

11

mode, and wherein said second network interface interleaves said primary mode encryption with said secondary mode encryption.

17. The medium of claim 16, in which a third network interface supports the protocol, said instructions including further instructions to direct said processor to:

balance processing said first network data into said second network data across said second and third network interfaces.

18. The medium of claim 17, wherein said balancing is performed according to a workload of said second and third network interfaces.

19. The medium of claim 15, wherein said first and second network interfaces operate in an adapter fault tolerance mode.

20. In a computing device, a network interface team, comprising:

a first network interface lacking hardware support for a protocol; and

a second network interface different from the first network interface, the second network interface including hardware supporting the protocol, said second network interface configured to transparently process network data for the first network interface if said network data is to be transmitted according to the protocol and to return processed data to the first network interface.

21. The network interface team of claim 20, further comprising:

a first receiver, communicatively coupled to said first network interface, for receiving network data to be transmitted by said first network interface;

a second receiver, communicatively coupled to said second network interface, for receiving network data to be transmitted by said second network interface; and

a transferor, communicatively coupled with said first network interface and said second receiver, and configured to transfer network data to said second network interface for processing according to the protocol.

22. A method for sharing processing capabilities of members of a system of network interfaces communicatively coupled with and operable to communicate over a network, comprising:

determining a first network interface is to transmit first data having a data configuration;

determining the first data is configured in accordance with a protocol unsupported by the first network interface;

locating a second network interface of the system including hardware that supports the data configuration;

12

transparently secondarily processing by the hardware of the second network interface of the first data in accordance with the protocol into a second data; and

providing the second data to the first network interface so that the second data appears to have been processed by the first network interface.

23. The method of claim 22, further comprising:

selecting the first network interface to transmit the first data based at least in part on a load-balancing of network traffic across the plural network interfaces;

performing by a driver for the first network interface of said determining the first data is configured according to the protocol unsupported by the first network interface;

receiving by the driver of the second data, wherein the data is now in a format supported by the network interface; and

providing by the driver of the second data to the first network interface.

24. A method for distributing network processing across a team of network interfaces cards including at least a first network interface card (NIC) lacking support for a first specialized capability and a second NIC that supports the first specialized capability, the method comprising:

receiving first data to be processed and transmitted by the first NIC to a recipient;

determining processing said received first data requires the first specialized capability unsupported by the first NIC;

transparently secondarily processing by the second NIC of the first data into second data with the supported first specialized capability; and

providing the second data to the first NIC for transmission by the first NIC to the recipient.

25. The method of claim 24, wherein the second NIC comprises an application specific integrated circuit providing the first specialized capability.

26. The method of claim 24, wherein the team of network interfaces include a third network that supports a second specialized capability, the method comprising:

aggregating specialized capabilities offered by interfaces of the team; and

providing a virtual NIC appearing to provide each of the specialized processing capabilities.

* * * * *

LAYER 4+ SWITCHING WITH QOS SUPPORT FOR RTP AND HTTP

Till Harbaum, Martina Zitterbart
Institute of Operating Systems and Computer Networks
Technical University of Braunschweig
Bültenweg 74, D38106 Braunschweig, Germany

Frederic Griffoul, Jürgen Röthig, Sibylle Schaller, Heinrich J. Stüttgen
Computer and Communication Research Laboratories
NEC Europe Ltd.
Adenauerplatz 6, D-69115 Heidelberg, Germany

Abstract

This paper provides an overview over current approaches and applications of layer 4 switching (L4Sw) and outlines a scheme for QoS support based on layer 4 and higher layer information. Today, L4Sw is mainly used for filtering in the context of firewalls. Additionally L4Sw has the potential of introducing per flow QoS support without the need for complex out-of-band signaling. We used traffic measurements on a real router to determine which higher layer protocols generate the most traffic and should therefore be investigated first. From the measurements it is obvious, that HTTP is the most important candidate. Experiments on extracting TCP and HTTP information demonstrated the feasibility of Layer4+QoS support with respect to processing power and storage capacity demands of intermediate systems. In addition to the TCP-based HTTP protocol, future investigations will be done for RTP as a key protocol for multimedia traffic.

Introduction

Traditionally, switching is a cell, frame or packet based network interconnection technique operating at layer 2 of the OSI reference model (e.g. ATM, Frame Relay and the like). This is in contrast to routing, which interconnects subnetworks at layer 3 of the OSI model. Generally speaking, a lower layer network interconnection (i.e. switching) performs more efficient than a higher layer interconnection (i.e. routing), whereas a higher layer interconnection provides better control and flexibility than a lower layer interconnection. In order to combine the benefits of both approaches, a new family of "integrated switch routers (ISRs)" a.k.a. IP switches has evolved over the last couple of years. However, recently there is an increasing demand for improved network services with regard to security, performance or QoS support for multimedia traffic. New switching techniques, using higher layer (Layer 4 and above) control information are being considered to provide these improvements. One of the main ideas behind this so called L4 switches (L4SWs) is, that higher layer information within the first packet(s) of a data flow can be utilized to set up a switched path for this flow. If this scheme can be extended to first detect and then to allocate the appropriate QoS for such a flow, we will have an attractive way of handling QoS allocation in-band, without

explicit and complex out-of-band signaling. However, this approach puts additional burden on the data forwarding path, which is highly performance critical. It is the main objective of this study to investigate, whether such a signaling-less QoS detection scheme is feasible and practical, i.e. whether the advantages of this scheme outweigh its disadvantages.

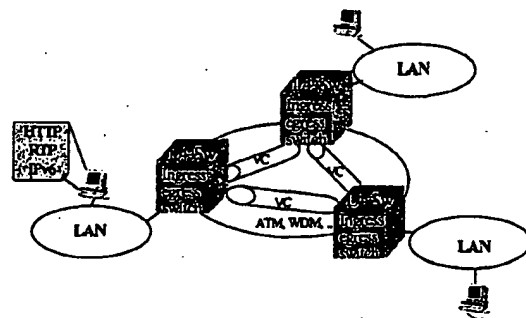


Figure 1: Layer 4 Switching Scenario

L4Sw is an approach that could be beneficially used at the edge node of a multi-gigabit IP backbone, independent of whether the backbone is based on ATM or other high performance transport technologies like WDM. High priority data or real-time flows may be detected as they leave the high-performance, typically over-provisioned LAN and enter the IP backbone, which generally requires explicit QoS support for performance and/or charging reasons.

A flow is defined as a number of related packets sent by applications over the network. Each flow has certain features. For instance, the download of content from a Web server which may be a collection of different data flowing between the server and a host, or the transmission of video or audio using special codecs.

Within this report, the use of layer 4 and above information to implement QoS support in IP backbones, is analyzed in detail. We briefly review the current state-of-the art in L4 switching and then identify the main technical challenges in the design of a QoS supporting L4 switching architecture, including an analysis of the implementation complexity of such an approach. Third we show traffic meas-

urements on an existing WAN ingress router, to identify the traffic flows that will actually be able to benefit from such an architecture.

Layer 4 Switching

Layer 4 switching uses transport layer information, e.g. TCP or UDP port numbers, to forward packets. Using layer 4 information to forward data allows to collect additional information about source and/or destination of a data frame. Not only the destination machine, but also the type of program transmitting and receiving the data units via well-known ports can in many cases be determined by the layer 4 information. Today's routers already use this information to enable or disable selected services (firewall).

Layer 4+ Switching

Layer 4+ switching (L4+Sw) describes the ability of a switch to look inside higher layer protocol information within a packet; i.e. L4+Sw accesses application header information for protocols built on top of TCP/UDP, like the HTTP or the RTP/RTCP headers.

In addition to layer 3 address information used for IP routing and layer 3 switching, layer 4+ switching uses information from the transport layer and above to forward data. The transport layer includes TCP and UDP and controls end-to-end communication between applications.

At layer 4 each TCP stream or UDP packet is associated with two port numbers. These port numbers identify the application protocol (FTP, HTTP, Telnet...) used with the corresponding communication. The protocol number is used at the IP layer to determine the appropriate transport protocol that has to handle the packet.

Some of these TCP/UDP port numbers have a determined meaning in all TCP/IP implementations [RFC1700]. The additional information supplied by the TCP/UDP port numbers is used for layer 4 switching.

Some protocols like RTP do not use well-known ports, other protocols like HTTP are not forced to always use their well-known ports. In these cases the port number itself does not help to determine the flow type. Additional information from layers above layer 4 (L4+) is needed to determine the flow type.

Applications and state-of-the-art

Several different approaches of L4Sw are currently used or discussed. They differ in the information extracted from the higher levels and the usage of this information. Today's main applications are:

Firewalls use layer 4 information inside the router for filtering. A firewall decides on the layer 4 port numbers whether information is forwarded or discarded for security reasons. Firewalls are easy to implement and consume only few additional CPU cycles inside a router to look up the

port number in the packet. This does not affect the routing tables of the router and, thus, can easily be added into existing routers. A firewall approach is discussed in [YAGO98].

QoS support based on layer 4 information uses the well-known port addresses to assign different service classes (priorities) to different data streams. To support QoS with layer 4+ information, the router has to extract content type information and the like from the data passing by in real-time. Classification based on well-known port numbers gives very coarse results. It fails completely for protocols not using well-known ports, e.g. RTP. For a fine grained classification further information from layer 4 and above is needed. This information can be used to determine the contents of for instance HTTP connections (text, graphics, audio, video...) and to detect and classify IP flows using protocols like RTP (audio, video, and whiteboard).

Current Layer 4 implementations

The existing implementations implement security filter and firewalls (YAGO, Foundry, Alteon). QoS support based L4Sw is mentioned in [BrPa97, YAGO98, Foun98], but no implementation or details are available.

Technical Challenges of L4+Sw

Adding layer 4+ support to the existing router and switch concepts may tighten some already existing bottlenecks of the packet forwarding process. Performance loss due to even more complex search and update operations may be the result. Other problems may arise due to the fact that the switch now needs to access the encapsulated data where classic routers and switches just don't need to care about the payload field and its encoding. The major technical challenge will presumably be the search engine using string comparisons to filter out HTTP parameters. Furthermore, the mapping of layer 4(+) based QoS parameters onto integrated/differentiated services has to be done by the switch. Another open challenge is the handling of encrypted data. Obviously encryption prevents the switch from interpreting the packet content. For a more detailed discussion see security section below.

Search Table

Depending on the kind of usage of layer 4+ information, the demands on CPU power and memory inside the switch can be very high [PeZu92]. Standard IP routers need to store information about all possible destinations [TaKZ94, ZHMB97, HMZB98, DBCP97, NiKa98]. This information may be compressed by using a net mask. In high-end routers, routing tables store up to 64k entries and a powerful search engine and algorithm is needed. With gigabit routing these tables quickly become system bottlenecks. Therefore, specialized hardware solutions are being developed to support scalable, fast and efficient table lookup. The hardware presented in [HMZB98] allows concurrent

FPGA based search, which is especially suited for multiple search algorithms needed with layer 4 switching as presented in [SVSW98]. According to [KeSh98] the problem of building fast and cost efficient table lookup support is a solved problem today.

Support for layer 4 port number handling increases the sizes of these search/routing tables dramatically, since for every entry additional information has to be stored. Standard IP routers need to hold one entry per destination address. Since L4+Sw handles different protocols separately, a L4+Sw needs to store one entry for each protocol used on each machine. As routing table lookup has already been a bottleneck for standard IP routers, fast lookup will become an even more important issue with L4Sw (Table 1).

	Standard IP router	Layer 4 switch
Table length	Number of destinations	Number of applications
Entry length	Length of IP address	Length of source and destination IP + port + additional information
Increased search complexity due to:	N/A	Less compression, longer tables, longer keys ...

Table 1: Router demands for Layer 4 switching

Furthermore the length of the keys is increased by the source address (which is not used in IP routers, but which is needed to identify a TCP connection) and the port numbers of source and destination and additional information extracted from layer 4 information.

The layer 4+ analysis and routing algorithm must be capable of handling keys of up to three times the length of keys used in standard IP routers. Also, the search engine has to cope efficiently with tables that are significantly larger than the routing tables of today's layer 3 switches and routers. A fast hardware implementation is needed to fulfill these demands.

Mapping of Traffic Parameters

The information derived from layer 4 and above needs to be mapped to the QoS parameters or classes of the respective network QoS model used, like IP differentiated services, IP integrated services, ATM, or other future QoS architectures.

This requires the isolation of traffic parameters and the interpretation of implicit service requests and priorities, given by the information extracted from layer 4 and above. In addition signaling for explicit reservations may be included to allow applications to reserve dedicated resources for a stream.

Security

With regard to secure communications the following needs to be considered: Is the data to be transmitted sensitive or confidential? Whom can I trust, and how much trust is appropriate? Depending on the answer to these and similar questions appropriate measures must be taken to protect the data during the transmission.

One such measure is encryption. But, encryption poses a problem to Layer4+ flow detection, because it may be impossible to access the necessary information without decrypting all or a certain part of the IP payload. However, it won't be done since CPU consumption and complexity are too high.

There are several options for Layer4+ flow detection in connection with security:

1. Don't use encryption.

This option, although best for Layer4+ flow detection and switching, is not acceptable in a lot of cases.

2. Encrypt at the ingress node into a L4+Sw cloud.

This allows for performing Layer4+ switching tasks and provides security over the switched path. This is viable only if the ingress node is inside the trusted part of the delivery path between the sender node and the first untrusted section towards the receiver. The algorithms and keys for encrypting packets must be known at the L4+Sw ingress node, which is feasible if the L4+Sw ingress node is inside the trusted region, e.g., located on the corporate premises. Also, the necessary resources for this additional task must be available (see Figure 2, option 2).

3. Explicit reservations for encrypted data streams.

A sender who wants to send encrypted data streams requests explicit reservations for a switched path towards the sender with the help of a reservation protocol, e.g. RSVP or ATM. This option does not use information derived from L4+Sw and may be used in combination with layer 4 based QoS.

4. Use of explicit flow identifiers like IPv6 Flow Labels

Here the data is encrypted at the sender. Packet classification is based on the Flow Label field of the IPv6 protocol (see Figure 2, option 3+4).

For option 1, no special effort is needed for Layer4+ switching. The interesting options are the options 2 and 4. When choosing option 2, encryption must be applied to the data after the flow detection test was done. For option 4 the first packet of a flow must be unencrypted, so that the QoS detection can work properly. All further packets of a flow may then be encrypted but can be classified properly based on the flow label.

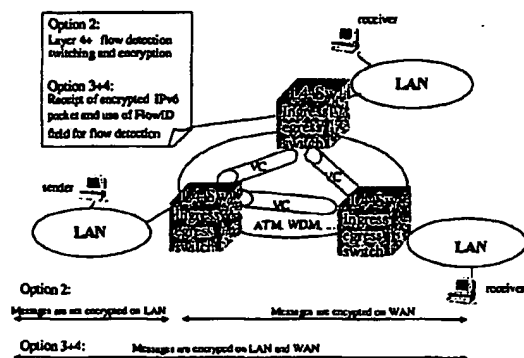


Figure 2: Encryption/Decryption on the Ingress Switch

TCP Traffic Measurements

To gain information about the distribution of different flow types to be expected, measurements on a router of the Technical University of Braunschweig have been performed. This router is in fact the ingress router from the institute network to the wide-area and the TU Braunschweig campus network, and therefore matches well the environment we are focusing on.

Basic scenario

The router used for the feasibility tests connects the Institute of Operating Systems and Computer Networks at the Technical University of Braunschweig with the internal campus net and the external Internet.

The local institute network consists of approximately 80 computers. These include local WWW and FTP servers, allowing us to monitor incoming HTTP and FTP requests. On the other hand, the internal network includes a workstation cluster for student lessons, generating a considerable amount of outgoing HTTP, FTP and other requests.

The router is running Linux, allowing modifications of the router software and direct access to the router network interfaces to capture the raw net traffic. To get this direct access, the network interface hardware was switched into promiscuous mode.

On the router a background process was used to monitor the passing traffic and to verify the L4+S_w algorithms. This process bypassed the TCP/IP processing within the Linux kernel. However, for the actual L4+ switching prototype, a kernel implementation has been developed for performance reasons.

Traffic monitoring

The first step of analysis was to take a look at the whole TCP traffic with respect to the port numbers in the TCP header.

Figure 3 shows the traffic monitoring over one week. It clearly shows, that most of the traffic is generated by HTTP and FTP. Other services using well-known ports like SMTP (mail), DNS (name resolution), X11 (X window system), SSH (secure shell) and NNTP (net news) generate little traffic. Some traffic is transferred over ports not linked to specific protocols or applications. This traffic includes for example students playing games over the net and experimental traffic, but it also contains HTTP traffic, that is not using the well known port for HTTP traffic (port 80). The results regarding HTTP and FTP match the results in [AMlz99, ThMW97]. These papers show an even higher amount of HTTP traffic.

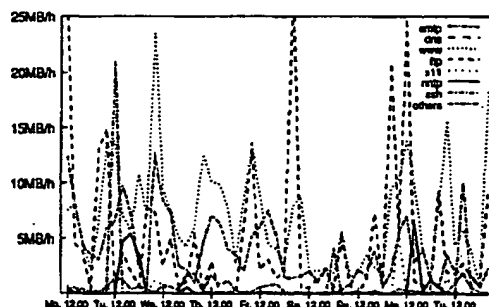


Figure 3: TCP/IP traffic monitored

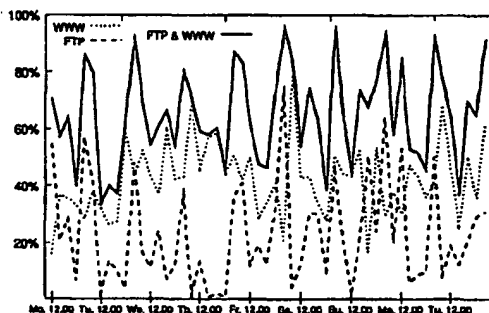


Figure 4: One week of FTP/WWW router traffic

While Figure 3 displays the absolute traffic in Megabytes per hour, Figure 4 shows the relative amount of FTP and WWW traffic compared to the overall TCP traffic. This figure clearly shows, that at least 40% of the overall traffic is either HTTP or FTP based. At most 93% was FTP or HTTP traffic and on average 70% was HTTP or FTP.

Due to the wide-spread use of the web for consumer applications, there will probably be an even higher amount of HTTP traffic on commercial routers operated by ISP's than the measurements from our university network show.

The TCP segmentation problem

A more detailed HTTP traffic analysis requires search engines to access the payload field inside the TCP transfer units, since HTTP is a TCP based protocol. Thus, TCP segmentation and reassembly inside the router will take place. Figure 5 shows an example of a TCP connection. In this example, the string 'This_is_a_demonstration' is transferred over a TCP/IP connection. Due to the internal TCP segmentation the string may be transmitted in small portions. In the example, the string is split into four TCP segments. If a search unit inside a router tries to find the string 'demonstration' in the passing traffic, it will never find the message, since there is no single packet containing the word 'demonstration'.

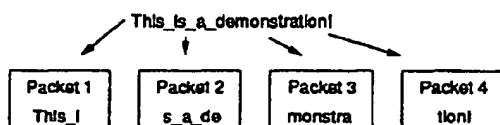


Figure 5: TCP segmentation

Therefore, for 100% error free HTTP analysis segmentation and reassembly has to be done for every stream to be analyzed inside the router. This means complex operations and up to 64k (the window size of TCP) buffer required for every TCP stream.

To get real life values on segmentation and reassembly, a TCP traffic analyzer was implemented on the router. This analyzer ran for about 30 hours. During this time all kinds of long and short distance connections were detected. There were low bandwidth connections as well as connections with a huge throughput and the internal servers were accessed from the outside as well.

The main results of these measurements are:

- The smallest single TCP segment contained 256 bytes payload. This segment size is big enough to hold the information needed for HTTP identification.
- All traffic was HTTP 1.0 based. The Results may vary with future HTTP 1.1 usage, because HTTP 1.1 allows aggregation of multiple HTTP transfers into one TCP connection.
- Over 99% of all HTTP headers could be identified as HTTP traffic by looking at one single TCP segment.

The most important result from this analysis is the fact that without performing a TCP reassembly inside the router over 99% of all HTTP traffic could be classified correctly. This allows fast and efficient HTTP/TCP processing inside the router without the demand of reassembly units and huge reassembly buffers and the additional delay.

HTTP traffic analysis

To get further information on HTTP handling, a HTTP analysis was started on the router. For layer4+ switching additional information from the HTTP header is used for classification and routing decisions.

HTTP header information of interest for layer 4+ switching may for example be:

- The name of the file being transmitted
- The type of the data transmitted (video, audio, text...)
- Source and destination of the data transmitted
- The expected transmission length

All this information may be included in the HTTP header, but most of this is optional. The measurements show that lots of connections don't use the Content Type field. Further investigation showed that the HTTP streams without valid Content Type could be identified by:

- The file name (.gif, .jpg, .html) contained in the preceding GET request from the client.
- If this file name was of unknown type, the content was in all cases plain HTML text.

All other traffic, especially high bandwidth connections for video and audio transmissions, had a valid Content Type field in the header of the HTTP reply (e.g. 'MPEG video' for MPEG-1 video transmission or 'RealAudio' for Real-Tek audio transmission).

This way the content type of all detected HTTP flows (i.e. 99% of all HTTP flows) could be determined.

Another interesting header field is the Content Length field. This field allows knowing already at the beginning of the transmission how many data follows. The measurements showed that 77% of all HTTP connections had a valid Content Length field. These connections were carrying about 87% of the overall HTTP traffic. This means that the transmissions without Content Length field are shorter than the average HTTP transmission. Other results from these measurements are:

- A connection transferred an average of 7590 bytes.
- 32% of all connections are shorter than 1000 bytes.
- The average amount of data transferred between two end systems was about 32900 bytes (giving an average of 4.3 connections per client server pair)

These measurements were done on the connection between the router and the external Internet and they exclude internal traffic between the institute and the campus network. During monitoring of the university's internal traffic a dramatic increase of the average amount of data transferred between two end systems was observed. This means that local HTTP connections last longer and the clients are more rarely 'hopping' between different servers. The aver-

age amount of data between two end systems was bigger than 300 kilobytes per session. This is ten times the value of the connections to the external Internet.

Implications

From the measurements presented in the last section we conclude, that the CPU and memory intensive segmentation is not really needed here. In the worst case, some HTTP flows may be missed and get the same treatment as they would without L4+ switching. Therefore HTTP analysis consists of two main steps:

step 1: detection and skipping of IP and TCP headers

Since this step only requires to extract both header length fields and to skip the complete IP and TCP headers by increasing the pointer to the payload field, this can be done with very low effort. This step is easy to implement and can easily be done in hardware.

step 2: search for 'magic strings' in payload

This is a complex task, requiring the search engine to step through the payload field and apply string matching functions. This step can be implemented in hardware, but requires a more complex hardware, than step 1.

The first step requires only a low effort and can easily be implemented in hardware. The more time and processing power consuming step is the second one which requires up to 200 comparisons for every single TCP packet. But since this information is only needed while the status of the connection is unknown this is only needed for very few packets (in average <5% of all packets need to be analyzed). However, a powerful hardware unit may be useful to support IP/TCP header skipping and fast pattern search in the payload field.

Conclusion

Layer 4+ information extracted from HTTP and other protocols like e.g. RTP traffic can be used for various applications like flow aggregation, reservation, prioritization.

First it can be applied to do packet classification for Differentiated Services, i.e. to support the Class of Service model for multimedia traffic.

Alternatively HTTP based layer 4 information can be used for bandwidth reservation. One obvious application is to reserve bandwidth for audio and video connections and to automatically route all audio and video traffic from a particular source through these reserved connections. Another application is to reserve bandwidth for dedicated client-server pairs and protocols.

Further, in the case of HTTP traffic, flow aggregation may be used to bundle several flows into one ATM VC or similar. It may be useful to bundle a couple of low traffic

connections or to bundle connections of the same type (one VC for HTML, one for graphics, one for audio...) or just to have one channel for all HTTP connections between a client-server pair.

Information for traffic handling may be derived from Layer 4+ information as discussed in this report. Although our first prototype is implemented in software, this function eventually requires hardware support, especially when used on a gigabit backbone network as intended.

References

- [Alte98] Alton Inc., Scaling Server Application Performance with Layer 4 Switching. TR March 1998.
- [AMlz99] J. Araci, D. Morato, M. Izal. Analysis of Internet Services in IP over ATM Networks, Proc. Of the 2nd Int. Conf. On ATM, Colmar, June 1999, pp.258-266
- [BrPa97] J. Bransky and L. Passmore. Layer 4 Switching; White Paper. Technical Report, Sept. 1997.
- [DBCP97] M. Degermark, A. Brodnik, S. Carlsson and S. Pink. Small Forwarding Tables for Fast Routing Lookups. In Proceedings ACM SIGCOMM '97, Cannes, France, Sept. 1997.
- [DoKN96] W. Döringer, G. Karjoth and M. Nassehi. Routing in Longest-Matching Prefixes. In IEEE/ACM Transactions on Networking, volume 4, Sept. 1996.
- [Foun98] Foundry Inc. Cutting Trough Layer 4 Hipe. Technical report, Jan. 1998.
- [HMZB98] T. Harbaum, D. Meier, M. Zitterbart, D. Brökelmann. Hardware Assist for IPv6 Routing Table Lookup. In SYBEN '98, Zurich, Switzerland, May 98.
- [NiKa98] S. Nilsson and G. Karlsson. Fast Address Lookup for Internet Routers. In Proc. IFIP/Broadband Communications, University of Stuttgart, April 1998.
- [PeZu92] T. Pei and C. Zukowski. Putting Routing Tables in Silicon. In IEEE Network Magazine, volume 6, IEEE, Jan. 1992.
- [RFC1700] J. Reynolds and J. Postel. ASSIGNED NUMBERS. RFC 1700, ISI, Oct. 1994.
- [SVSW98] V. Srinivasan, G. Varghese, S. Suri, M. Waldvogel. Fast and Scalable Layer 4 Switching, July 1998.
- [TaKZ94] A. Tantawy, O. Koufopavlou and M. Zitterbart. On the Design of a Multigigabit Router. In Journal on High Speed Networks, volume 3, 1994.
- [ThMW97] K. Thompson, G. J. Miller and R. Wilder, Wide-Area Internet Traffic Patterns and Characteristics, IEEE network, vol. 11, no.6, December 1997
- [Yago98] YAGO Inc. Layer 4 Switching: An Overview. Technical report, March 1998.
- [ZHMB97] M. Zitterbart, T. Harbaum, D. Meier and D. Brökelmann. Efficient Routing Table Lookup for IPv6. IEEE HPCS '97 Workshop, Chalkidiki, Greece, 6/97

Optimizing TCP Forwarder Performance

Oliver Spatscheck, Jørgen S. Hansen, *Student Member, IEEE*, John H. Hartman, *Member, IEEE*, and Larry L. Peterson, *Senior Member, IEEE*

Abstract—A TCP forwarder is a network node that establishes and forwards data between a pair of TCP connections. An example of a TCP forwarder is a firewall that places a proxy between a TCP connection to an external host and a TCP connection to an internal host, controlling access to a resource on the internal host. Once the proxy approves the access, it simply forwards data from one connection to the other. We use the term *TCP forwarding* to describe indirect TCP communication via a proxy in general. This paper briefly characterizes the behavior of TCP forwarding, and illustrates the role TCP forwarding plays in common network services like firewalls and HTTP proxies. We then introduce an optimization technique, called *connection splicing*, that can be applied to a TCP forwarder, and report the results of a performance study designed to evaluate its impact. Connection splicing improves TCP forwarding performance by a factor of two to four, making it competitive with IP router performance on the same hardware.

Index Terms—Firewall, proxy, router, TCP.

I. INTRODUCTION

IT IS increasingly common that processes communicate with each other indirectly through a proxy. This happens, for example, in a firewall where a proxy mediates the flow of information between a TCP connection to an untrusted external entity and a TCP connection to a trusted local entity. We use the term *TCP forwarding* to denote the general pattern of indirect communication over a pair of TCP connections via a proxy.

One consequence of TCP forwarding is that there is often a single network node—e.g., a firewall—that runs proxies on behalf of many different indirect communications. This network node, which we call a *TCP forwarder*, plays a role very similar to that of an IP router, except it must execute two TCP endpoints and a proxy for every “flow” that passes through it. To intercept the TCP connections successfully it has to receive *all* TCP packets for both TCP connections. This can be achieved by either addressing the TCP forwarder directly or by placing it on a choke point in the network. Therefore, the performance of the TCP forwarder, i.e., its throughput in terms of packets-per-second, can play a significant role in the network performance perceived by the communicating entities.

Manuscript received May 15, 1998; approved by IEEE/ACM TRANSACTIONS ON NETWORKING Editor S. Pink. This work was supported in part by Defense Advanced Research Projects Agency Contract DABT63-95-C-0075 and Contract N66001-96-8518, and by National Science Foundation under Grant CCR-9415932.

O. Spatscheck is with AT&T Labs—Research, Florham Park, NJ 07932-0971 USA (e-mail: spatsch@research.att.com).

J. S. Hansen is with the Department of Computer Science, University of Copenhagen, Copenhagen, Denmark (e-mail: cyller@di.ku.dk).

J. H. Hartman is with the Department of Computer Science, University of Arizona, Tucson, AZ 85721 USA (e-mail: jhh@cs.arizona.edu).

L. L. Peterson is with the Department of Computer Science, Princeton University, Princeton, NJ 08544 USA (e-mail: llp@cs.princeton.edu).

Publisher Item Identifier S 1063-6692(00)03322-7.

This paper makes three contributions. First, it defines briefly a general framework for TCP forwarding, and demonstrates the relevance of TCP forwarding to three applications: firewalls, HTTP proxies, and mobile computing. Second, it describes an optimization technique, called *connection splicing*, that can be used to improve the performance of a TCP forwarder. An implementation of connection splicing in the Scout operating system is also presented. Third, it reports the results of a performance study that measures the effectiveness of connection splicing. The study shows that connection splicing improves TCP forwarder performance by a factor of two to four, bringing its performance close to that of an IP router implemented on the same platform.

II. TCP FORWARDING

When two entities communicate indirectly through two separate TCP connections, an entity called a *proxy* mediates the communication, interposed between the two connections, and controls the flow of data between the communicating parties. The proxy decides if the parties can communicate, and if so, what is communicated. A proxy can both restrict and enhance the communication. For example, a Telnet proxy can restrict to which computers the outside world may connect, and perhaps which users may log in. On the other hand, a Telnet proxy could also serve as a clearinghouse for a collection of servers by providing a single connection point for outside Telnet accesses. The Telnet requests are processed by the proxy and forwarded to the appropriate computer, shielding the outside world from the internal structure of the site.

We use the term *TCP forwarding* to refer to communication relayed over two TCP connections via a proxy. TCP forwarding is not as simple as copying bytes from one connection to the other, however. The proxy must control the communication as well as relay bytes, and therefore, a proxy has two modes: *control mode* and *forwarding mode*. In control mode the proxy processes either out-of-band or in-band control information. Once the control functions have been completed, the proxy switches to forwarding mode to move data between the connections. After the data transfer, the proxy may switch back to control mode. For example, a Telnet proxy starts off in control mode, and processes a Telnet request to determine if the connection should be allowed, based on the target machine, port, and perhaps user ID. Once the connection has been completed, the proxy switches into forwarding mode to transfer data between the two computers. Switching between these two modes of operation is the primary difficulty in developing an optimized TCP forwarding mechanism.

The processing done in control mode varies greatly between proxies, ranging from very little processing during connection

setup, to continuous monitoring of the data stream while forwarding to extract control information. Proxies can be broadly classified into four categories, depending on the degree of control processing they do.

The first class of proxies perform a minimum of control processing; they typically perform level-4 routing based on IP addresses and port numbers. They are in control mode only during connection setup, after which they switch to forwarding mode for the duration of the connection. An FTP proxy is an example: it processes an FTP request in control mode on the control connection, sets up a data connection between the two computers, and switches to forwarding mode on the data connection until it is closed. The control connection remains in control mode to process subsequent FTP requests.

The second class of proxies performs more control processing because they authenticate the user or request and base routing decisions on either the result of the authentication or control information passed in the TCP connection. A Telnet proxy is a member of this class. Typically, a Telnet proxy requests a user ID, password, and the destination of the Telnet request. This information is received on the TCP connection by the proxy and is used to authenticate the user and establish a connection to the correct remote machine. At this point, the proxy simply forwards data between the two connections.

The third class of proxies remains in control mode for all data transferred in one direction, but switch to forwarding mode for data transferred in the other. An example is an HTTP proxy that processes the HTTP requests (control information) sent by clients, but simply forwards the data returned by the HTTP server.

The fourth class remains in control mode and continuously monitors data passed in both directions. This might be the case for a proxy that allows users on a protected network to access HTTP servers on the Internet. The proxy could filter outgoing accesses to restrict the servers that can be reached, and filter incoming access responses to remove (untrusted) Java code.

TCP forwarding has many uses, including such diverse functions as a network firewall, an HTTP proxy, and a mobile computing system. These three examples illustrate the power of TCP forwarding, and motivate the need for an efficient implementation.

A. Firewall

A firewall provides limited connectivity between a protected network and the relative chaos of the Internet, as shown in Fig. 1. The firewall contains different types of proxies, each handling a different type of communication between the two networks, such as Telnet, FTP, etc. A typical proxy accepts connections on one network, authenticates the entity making the connection request, and forwards the data to the other network, perhaps after applying a filter. The firewall either uses its own IP address (*classical proxy*) or is completely transparent to the user (*transparent proxy*) [6]. A classical proxy must use the control information in the request to determine the connection's true destination.

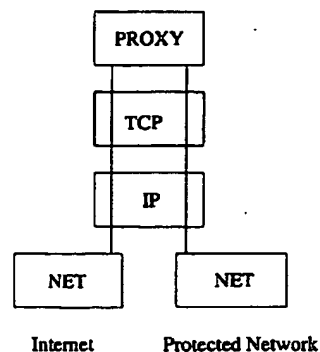


Fig. 1. Overview of an application-level firewall. Data from one network pass through the proxy which forwards them to the other network if the desired security guarantees are not violated.

B. HTTP Server Proxy

TCP forwarding can also be used to develop scalable servers such as HTTP servers. HTTP server names are embedded in the URL namespace, making it difficult to implement a single HTTP service from a collection of servers. Load-balancing across the collection is a problem; web sites typically offer the users a selection of servers from which to choose, manipulate the DNS mappings to change dynamically the IP address associated with a site name, or use the HTTP redirection mechanism to redirect requests to unloaded servers. The first two offer coarse-grained load balancing, while the last requires two HTTP connections per URL accessed.

An HTTP server proxy that forwards TCP connections is a better solution. Clients connect to the proxy, which processes their requests and forwards them to the appropriate server. The proxy must continually monitor the data received from the clients, however, so that requests can be extracted and processed, and the connections re-forwarded as appropriate. The data returned from the servers, however, is simply forwarded to the clients.

Such an HTTP proxy might implement a variety of forwarding policies, in addition to load-balancing over a set of homogeneous servers. The proxy could forward connections to servers based on the URL requested, allowing a collection of servers, each of which serves a different collection of pages, to appear as a single site. It could also provide more complex functionalities as described by Brooks *et al.* [5].

C. Mobile Computing

Our final example involving proxies is from the area of mobile computing. Here proxies are used to improve the performance of mobile hosts operating across wireless links by separating TCP connections into two connections; one covering the wireless link and one covering the wired network. The performance enhancement can either be simply an improvement caused by the separation of flow control on the two different types of network, or it can rely on transformation or filtering of data, e.g., the proxy reduces the resolution on graphics sent to the mobile host over a low capacity link and removes all video clips from e-mail. The situation is complicated by the fact that mobile hosts often use a mixture of wireless and wired networks, switching between them on the fly. When the mobile host is con-

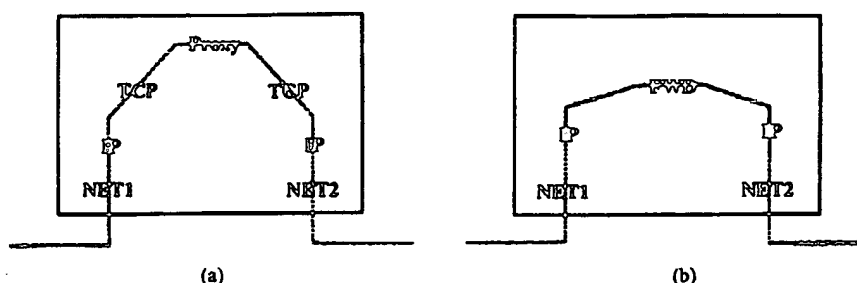


Fig. 2. Optimizing two TCP connections into a single spliced connection. (a) Unoptimized TCP forwarder. (b) Optimized TCP forwarder (with spliced connection).

nected to a wired network, the proxy merely relays data in the forwarding mode, but cannot be removed from the path of communication due to the presence of the bipartite TCP connections.

Another use of proxies is to allow a mobile host to change its point of attachment to the network without jeopardizing any open connections. In this case the proxy would operate in the forwarding mode when the mobile host is connected, but would switch to control mode both when the mobile host connects and when it disconnects. This would allow the mobile host to terminate its TCP connections, move to a new location with a new IP address, and establish a new set of TCP connections to the proxy without affecting the peer hosts on the other side of the proxy.

III. CONNECTION SPLICING

This section describes an optimization technique, called *connection splicing*, that improves TCP forwarding performance. It includes a discussion of the many complications that make connection splicing difficult in practice. To simplify the following discussion, we focus on the flow of data in a single direction; the same work must also be done for data going in the other direction.

A. Overview

The proxy involved in TCP forwarding operates in either control mode or forwarding mode. The basic idea of connection splicing is to detect when a proxy makes a transition from control mode to forwarding mode, and then splice the two TCP connections together into a single forwarding path through the system. The resulting spliced connection replaces the processing steps (and associated state) required by two TCP connections with a single reduced processing step (and associated state).

Fig. 2 schematically depicts the optimization. The standard (unoptimized) forwarder on the left requires TCP segments to traverse TCP twice, with each instance of TCP maintaining the full state of the connection. In this case, the proxy simply passes segments from one connection to the other when it is in forwarding mode. The optimized forwarder on the right replaces the proxy and two TCP processing steps with a single FWD processing step. FWD maintains just enough state to forward TCP segments successfully from one network to another. The state FWD needs to maintain is described later in this section.

A single proxy might require both configurations, however.

The configuration on the left *must* exist when the proxy is in

control mode; the proxy must be in the loop because it needs to inspect the data flowing between the two TCP connections. The configuration on the right *may* exist while the proxy is in forwarding mode. Forwarding can also happen in the left configuration, but performance suffers. With this perspective in mind, there are three cases to consider: how the optimized configuration on the right works in the steady state (Section III-B), how the system makes the transition from the left-hand configuration to the right-hand configuration (Section III-C), and how the system makes the transition from the right-hand configuration back to the left-hand configuration (Section III-D).

Typically, TCP forwarding starts in the unoptimized configuration, makes a transition to the optimized configuration when the proxy shifts from control to forwarding mode, and sometimes reverts back to the unoptimized configuration should TCP forwarding go back to control mode. Note that while the connection splicing optimization is in effect, the two independent TCP connections shown on the left no longer exist on the forwarder.

B. Forwarding

The primary task of the FWD processing step shown in Fig. 2 is to change the header of incoming TCP segments to account for the differences in the two original TCP connections. Since the two TCP connections were established independently, their respective port numbers and sequence numbers are probably different. The IP addresses associated with the connections might also differ, resulting in changes that affect the IP pseudo header as well.

Fig. 3 depicts the TCP segment header; the boldface fields are those that FWD modifies. The following outlines the transformations FWD applies to each segment it forward from one connection (*A*) to another connection (*B*). For now, we ignore the problem of moving a TCP forwarder into the optimized state, and focus instead on the work involved in forwarding segments once FWD is in place. Also, we assume that the two TCP connections were established independently. If their establishment was in fact interleaved—so that one connection knew what port and sequence numbers were being used by the other connection—then additional optimizations are possible, as described in Section III-F.

- **Port Numbers:** If the TCP forwarder operates as a classical proxy, the port numbers of both TCP connections will probably differ. Therefore, the source and destination port numbers of segments arriving on *A* have to be changed to the port numbers of connection *B*. If the TCP forwarder

SrcPort		DstPort	
SeqNum			
Ack			
Hlen	Resv	Flags	AdvWin
CksNum		UrgPtr	
Options		Padding	
Data			

Fig. 3. TCP segment header with fields modified by FWD in bold.

is a transparent proxy, this change is unnecessary because the proxy uses the same port numbers as the initiator.

- **Sequence Number:** The sequence number used by segments received by FWD on *A* are probably different from those used for segments sent by FWD on *B*. This is because TCP initializes sequence numbers randomly for each independent connection. The sequence number for an outgoing segment is computed by adding a fixed offset to the sequence number in the incoming segment.
- **Acknowledgment Number:** The acknowledgment number acknowledges the sequence numbers forwarded *in the other direction*. Thus the acknowledgment number in an outgoing segment is computed by subtracting from the sequence number in the incoming segment the sequence number offset for segments flowing in the other direction.
- **Checksum:** Modifying the other fields requires adjusting the TCP checksum. A constant checksum patch representing the “delta” in the checksum is used to do this efficiently. If the FWD acts as a classical proxy, the changes to the IP address fields in the IP pseudo-header are also reflected in this checksum patch.

The following pseudo code describes the changes to a segment transferred from *A* to *B*. All header fields marked Input represent the segment header values in the received segment. The header fields marked Output represent the segment header values used in the outgoing segment. Bold variables indicate constants that are part of FWD’s state. Subscripts indicate the direction for which these constants are used; e.g., $\text{SeqNumOffset}_{A \rightarrow B}$ represents the sequence number offset used to patch sequence numbers on segments received from *A* and sent to *B*.

```

Output.DstPort = RemotePortB
Output.SrcPort = LocalPortB
Output.SeqNum = Input.SeqNum + SeqNumOffsetA→B
Output.Ack = Input.Ack - SeqNumOffsetB→A
Output.Cksum = Input.Cksum + CksumPatchA→B.

```

The checksum calculation shown in the pseudo code is more complicated than simple addition. To account for overflows or underflows during sequence number and acknowledgment

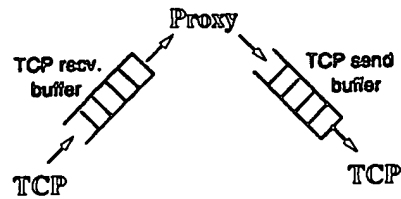


Fig. 4. TCP buffers potentially containing acknowledged data.

number calculations it is necessary to add or subtract one from the checksum. This is because the checksum is the one’s complement of the one’s complement sum of the segment.

Splicing two TCP connections significantly changes the behavior of the forwarding proxy. In the unspliced case segments sent to the proxy are acknowledged when they are processed by the incoming TCP stack. The proxy then takes responsibility for the data, resending them as necessary to ensure they reach their destination. Data are buffered in the outgoing TCP stack until they are acknowledged by the destination. When the two connections are spliced the segments no longer traverse the two TCP protocol stacks. The proxy doesn’t acknowledge data coming from the sender, nor does it resend data to the destination. Data and acknowledgments are forwarded without processing, requiring the two endpoints to handle retransmission and reordering.

Forwarding segments requires internal state in FWD. Some of this state is required to modify the header fields, such as the port numbers, sequence offsets, and checksum patch. FWD must also detect reset or termination of the TCP connection. To do so it parses the flags in the header and keeps a simplified TCP state machine. FWD also keeps one timer that is used to time-out the connections; all other TCP timers are not used.

If it is possible that the optimized forwarder will revert back to a standard forwarder, FWD also needs to store the current advertised window, the highest sequence number sent, and the highest ACK seen that will fit in the advertised window. The process of converting a spliced connection back to an unoptimized TCP forwarder is discussed in Section III-D.

C. Splicing

The header modifications required to forward a segment are relatively straight-forward. The real problem is transitioning from the unspliced state to the spliced state. The difficulty is caused by acknowledged data buffered in the forwarder. This data might be buffered by the receiving TCP’s receive buffer, within the proxy itself, and in the sending TCP’s send buffer (Fig. 4). The acknowledged data must be reliably forwarded to its destination. These data also influence the offsets calculations required by the spliced connection.

First, all data acknowledged by either connection on the unoptimized TCP forwarder must be reliably delivered to their destination. The important point is that these data have already been acknowledged by the forwarder, so it cannot depend on the source host to take responsibility for possibly retransmitting the data in the future. Thus, the forwarder must continue to run TCP—the only way it can reliably deliver data—until the currently buffered (acknowledged) data are reliably delivered to

the destination. During the time the data are being drained, however, new segments may arrive. The forwarder obviously cannot let TCP acknowledge the new data, because doing so will just give it even more data to deliver reliably, and it is impractical to wait until the two connections go idle before completing the splice. Fortunately, there are two ways to handle newly arriving segments during this transition period.

The first option is to delay the activation of the spliced connection until after the buffers have drained. During this time, the limited number of new segments that arrive are delivered to TCP so that any acknowledgment they carry can be processed, and then they are held in a separate buffer for FWD. These incoming segments are not themselves acknowledged and they are not placed in the incoming connection's receive buffer. If the buffer overflows while TCP is still processing acknowledgments, the segments are dropped after the acknowledgments have been processed. When the transition is complete, these buffered segments are processed by FWD as though they had just arrived. Again the TCP protocols are suspended as soon as all buffers are drained. This solution may drop data if the FWD buffers overflow while the TCP buffers are being drained. If the amount of data buffered in TCP is small, then the FWD buffers are unlikely to overflow.

The second option allows FWD to begin forwarding data concurrently with draining the buffers. Should any new data arrive during the transition, it is important that the original TCP protocols do not acknowledge the new data; they are only allowed to process the acknowledgments contained in those segments so that the buffers drain. In other words, all newly arriving segments are delivered to both the original TCP protocol (for acknowledgment processing only) and to FWD (for forwarding to the receiver). This solution does not drop data, but may cause data to be delivered out-of-order. This is because segments processed by FWD may be delivered before segments traversing the original TCP connections. This will not affect correctness because the destination will reorder the segments.

During the time that FWD operates concurrently with the draining process, both forwarded segments and drained segments will arrive at the destination. This means it is possible that the TCP draining buffers on the forwarder might receive an acknowledgment for a sequence number that is larger than the maximum sequence number in its send buffer. This acknowledgment is meant for both the source host and the TCP forwarder. It is most likely due to dropped ACK's or delayed ACK processing on the receiver. Since the forwarder is still processing acknowledgment in an attempt to drain its buffers, it will receive this acknowledgment too. To allow for this possibility, TCP running on the forwarder during the transition must be able to accept acknowledgments up to one full window size larger than the maximum sequence number in its send buffer.

Before the packet processing can be altered, the internal state of FWD has to be initialized, corresponding to the first step above. This requires computing the sequence number offsets ($\text{SeqNumOffset}_{A \rightarrow B}$ and $\text{SeqNumOffset}_{B \rightarrow A}$) and the checksum patches ($\text{CksumPatch}_{A \rightarrow B}$ and $\text{CksumPatch}_{B \rightarrow A}$) used by FWD. The sequence number offsets can be calculated as soon as all acknowledged data have been drained. If acknowledged data still exist in one of the

forwarder's buffers, then it is necessary to subtract the length of the buffered data from the corresponding sequence number offset. This is because the sender of a segment that is directly forwarded assumes that the buffered data were delivered, and therefore, the sequence number of the source's TCP protocol has already been increased. It is important to realize that the sequence number offset cannot be calculated earlier since we do not assume that the proxy will forward all data or add no additional data to the data stream while it is in control mode. The checksum patch can be calculated as soon as the other offsets are known since the changes in port number and IP address are already known.

D. Unsplicing

When the forwarding proxy switches from forwarding mode to control mode the connections must be unspliced. There are two complications. The first is to be able to detect that it is necessary to switch back to the unoptimized state; i.e., that the forwarder has moved from forwarding mode to control mode. The second is to correctly make the transition. The solutions to these two complications are intertwined.

It may be difficult to decide when the proxy should switch back to control mode. If the control information is sent over the spliced connection, the proxy has to monitor the data being forwarded to detect the control information. This is difficult because the FWD protocol does not reorder the segments it receives, nor does it buffer segments. The proxy has to find the control information by looking at out-of-order segments, one at a time. This makes it unlikely that the proxy will be able to filter the data to find control information. However, it seems useful to trigger a switch back to unoptimized mode as soon as data are transmitted in a certain direction. An HTTP 1.1 proxy, for example, might allow the forwarding of HTTP replies, but want to examine all (possibly pipelined) HTTP 1.1 requests. The cost of detecting this switch could vary greatly, ranging from simple monitoring if data flows in a certain direction for HTTP 1.1—which can be done by comparing a single sequence number—to maintaining a shadow state machine of the higher level protocol.

Dealing with acknowledgments makes it difficult to unsplice a connection. When the forwarder reverts to two TCP connections, it must take over handling acknowledgments. If there are no unacknowledged segments outstanding on the spliced connection, the transition back to unspliced is easy. The reconstructed TCP connections are initialized with the sequence numbers, acknowledgment numbers, and advertised window sizes stored as FWD state. The state-machine is progressed to the current state, the timers and the send window are initialized with their initial values, and a slow start is initiated. The slow start is necessary since no bandwidth estimates are available, and therefore, the congestion window sizes have to be rediscovered. In the case where no unacknowledged segments are outstanding, it is possible to stop forwarding new segments instantly.

If there are outstanding unacknowledged segments, however, the forwarder must either wait for all of them to be acknowledged—dropping data if necessary—and then switch as described above, or else it must continuously monitor the

segment stream until it has copies of all unacknowledged segments. It then uses this information to initialize the TCP connections and buffers. This solution does not drop any segments, but up to two full window sizes might have to be buffered before the switch over can be completed.

E. Flow Control

During unoptimized operation flow control is handled by the two independent TCP protocols on the forwarder, and the TCP protocol on the end hosts. During optimized operation, flow control is handled by the end hosts only; the forwarder merely drops segments, just as a congested router drops IP datagrams.

There is a complication during the transition to a spliced connection, however. Shortly after the switch to the spliced connection, the advertised window might be either too big or too small. For example, the window advertised by the destination host to the forwarder might be smaller than the window advertised by the forwarder to the source host. In this case, the source host will suddenly see a smaller advertised window after the connection is spliced, possibly triggering unnecessary retransmissions. Similarly, the send window of a host might also be bigger than the advertised window of its new peer. If so, it is likely that data will be transmitted unnecessarily. Note that RFC1122 strongly recommends¹ that the advertised window not be reduced to eliminate this unnecessary data transmission. To minimize this problem, the TCP forwarder can restrict the size of the window it advertises to the source host to the window size advertised by the destination host, minus the size of the buffered data.

More subtly, the send window of both end hosts might not represent the bandwidth of the link. If the send window is too big, the host will send too many segments and generate unnecessary congestion. However, this can only happen if traffic is extremely bursty. Otherwise, the limited buffer space available on the TCP forwarder should synchronize the send window sizes of both TCP connections.

Another issue is the increase in end-to-end round trip time (RTT) after a pair of connection have been spliced. In general, TCP's throughput decreases if the RTT increases. This is due to either an increased $RTT \times \text{bandwidth}$ product that is greater than the advertised window, or a slower increase in the sender's congestion window during the slow start or congestion avoidance phases. In either case, there are no effects on the RTT that would not have been present had the connection run end-to-end in the first place.

F. Additional Optimizations

The connection splicing optimization can be applied not only at the TCP level, but also to unfragmented IP datagrams. In addition, the optimization can be applied to the first IP fragment of an IP datagram if we allow the unfiltered forwarding of all remaining fragments, and if the MTU is large enough so that the first fragment will contain the TCP segment header.

In these two cases, the forwarder can process the IP datagrams similarly to an IP router, with the additional TCP segment header manipulation described in the previous section. Fig. 5 illustrates this scenario, which we denote as a combined IP/FWD

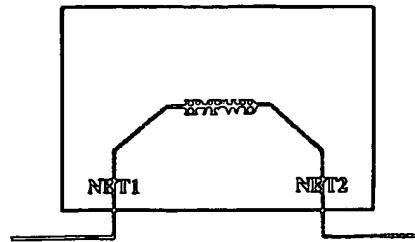


Fig. 5. FWD merged with IP. Further optimizing the spliced connection when there is no fragmentation.

processing step. The important consequence of being able to forward TCP segments at the IP level is that it makes it possible to apply any of the optimizations one might apply to an IP router. For example, if the forwarder is connected to two Ethernets, we can modify and forward the Ethernet packets directly.

Finally, under certain circumstances it is possible that the TCP forwarder can tolerate the unfiltered forwarding of all IP fragments, that is, FWD implements the identity transformation. This would happen if the unoptimized forwarder is configured as a gateway intercepting TCP connections and was careful in selecting port numbers and the starting sequence numbers when the original pair of TCP connections were opened. This being the case, FWD can be omitted and the TCP forwarder operates just like an IP router.

G. Other Issues

Additional IP-level filtering can also be done on a spliced connection. For example, to avoid attacks against the TCP stack the forwarder should limit the sequence and acknowledgment numbers of the current spliced connection to meaningful values, and drop all segments that have the SYN flag set. This is possible since all connections are established first with the proxy, not with the final destination.

As many routers already do, the forwarder can also perform network address translation (NAT). It is possible to perform NAT on spliced and unspliced connections. If, however, IP addresses are passed within the data stream, as in FTP for example, the connection has to either be spliced after the IP addresses have been altered by the proxy, or additional IP-level filters have to be added.

A final issue is TCP options. Our prototype currently supports only the MSS option, which is negotiated with the forwarder. If the MSS of both segments do not match, the ICMP Destination Unreachable message will be used to adjust the MSS after the connection is spliced. The other TCP options can be handled in much the same way as the prototype patches sequence numbers; some (e.g., SACK) don't require additional state, but others (e.g., TIMESTAMP) do.

IV. CONNECTION SPLICING IN SCOUT

Connection splicing can be implemented in any operating system; Section VI discusses an implementation in Unix. This section describes an implementation in an OS designed specifically to support communication: Scout [12]. While the primary purpose of this section is to flesh out some of the details any

¹In IETF terminology, the operative word is **SHOULD**.
Appellant's Brief ER 163796141 US

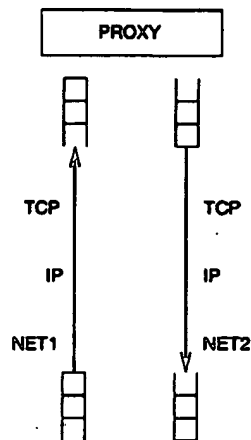


Fig. 6. TCP forwarding implemented in two Scout paths.

implementation would have to address, it has a secondary objective of illustrating how naturally a technique like connection splicing can be realized in an operating system designed around communication-oriented abstractions.

Scout is a configurable OS explicitly designed to support data flows, such as video streams through an MPEG player, or a pair of TCP connections through a firewall. Specifically, Scout defines a *path* abstraction that encapsulates data as they move through the system, for example, from input device to output device. In effect, a Scout path is an extension of a network connection through the OS. Each path is an object that encapsulates two important elements: 1) it defines the sequence of code modules that are applied to the data as they move through the system, and 2) it represents the entity that is scheduled for execution.

The path abstraction lends itself to a natural implementation of TCP forwarding. Fig. 6 schematically depicts a naive implementation of TCP forwarding (the unoptimized case) in Scout. It consists of two paths: one connecting the first network interface to the proxy and another connecting the proxy to a second network interface. In this figure, the path has a source and a sink queue, and is labeled with the sequence of software modules that define how the path “transforms” the data it carries.² To a first approximation, the configuration of Scout shown in Fig. 6 represents the implementation one would expect in a traditional OS.

The two-path configuration shown in Fig. 6 has suboptimal performance because it requires a handoff of each incoming segment from the first path to the proxy, and then from the proxy to the second path. In Scout, the entire device-to-device data flow can be encapsulated in a single path (Fig. 7). This is the implementation of choice for the unoptimized TCP forwarding case in Scout.

Connection splicing is then implemented within the same framework. Fig. 8 illustrates the two optimized configurations discussed in Section III: the path on the left corresponds to the right-hand case from Fig. 2, while the path on the right corresponds to the case shown in Fig. 5. Note that the right-hand path looks very much like an IP router would in Scout.

²As in Section III, we focus on data flowing in one direction. In reality, Scout paths, like TCP, supports bidirectional data flows.

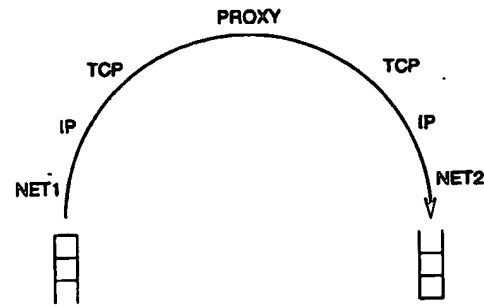


Fig. 7. TCP forwarding implemented in a single Scout path.

Looking at the implementation in a bit more detail, each path consists of a linked list of *stages*, where each module that the path traverses contributes a stage to the path during path creation. Abstractly, each stage contains the path-specific code and state for the corresponding module; e.g., the TCP control block is contained in the TCP stage of the paths shown in Figs. 6 and 7. When the proxy in an unoptimized TCP forwarding path detects a transition to forwarding mode, it does five things:

- Stops processing incoming segments and allows segments to accumulate in the path’s input queue.
- Unlinks the two TCP stages and the proxy stage from the path and replaces them with a preliminary FWD stage.
- Continues processing incoming segments and data in the TCP buffers until the TCP buffers are drained.
- Unlinks the preliminary FWD stage and replaces it with the final FWD stage.
- Continues processing incoming segments.

The difference between the preliminary FWD stage and the final FWD stage is that the former forwards the segments and reliably drains the TCP buffers, whereas the latter only adjusts segment header fields.

One subtlety is that there are seldom any segments queued *within* the path that need to be drained: Scout is nonpreemptable, so in practice once a segment is removed from the input queue it is processed completely and deposited in the output queue. The only time a segment gets buffered in the middle of a path is when the scheduler selects the path for execution, the segment makes it as far as the outgoing TCP stage, but the advertised window on the second connection is closed. It would be possible to take the outgoing window into account when making the scheduling decision—i.e., not schedule a TCP forwarding path until it was certain that the segment could make it all the way to the output queue—but the consequence is that the segment would remain in the input queue, and thus, not acknowledged on the incoming TCP connection.

There is one final issue to consider: how Scout classifies each incoming packet to determine the path to which it belongs. Scout classifies packets by inspecting various header fields, such as ETH’s type field, IP’s protocol field, and TCP’s port fields. While the details are beyond the scope of this paper, the relevance to connection splicing is that even after an unoptimized TCP forwarding path is spliced, the classification machinery remains the same. In other words, the spliced path no longer does any TCP processing, but the TCP port fields are still used to classify packets for the spliced path.

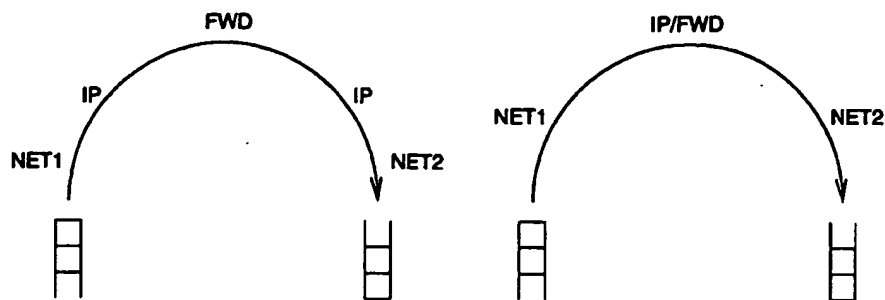


Fig. 8. Connection spliced paths in Scout.

V. PERFORMANCE

This section provides measurements of the effect of connection splicing on TCP forwarding. To make the study concrete—and to give us an existing system against which we can compare our approach—we focus on a simple firewall configuration. The proxy in the firewall does not perform any processing in control mode; it is always in forwarding mode.

A. Test Cases

We measured the following configurations of Scout:

- **2-Path:** This is a full blown TCP forwarder, as depicted in Fig. 6. This TCP forwarder uses two separate TCP paths, containing two entirely independent TCP state machines, that meet at the proxy—one to each network device. As going from one path to another often will require a context switch, this configuration is the closest to the structure of a firewall in a regular operating system like Unix or NT.
- **1-Path:** This is the configuration shown in Fig. 7. This case is similar to the 2-path configuration, except the two network devices are connected by a single path. This is the natural way of expressing a TCP forwarder in Scout. Note that this configuration still involves two TCP connections implemented by two independent TCP state machines but a single Scout path.
- **FWD:** This is an optimized version of 1-Path. Here the TCP connections have been spliced into a single connection, and the forwarder is reduced to updating the TCP headers. This configuration still supports reassembly of IP packets. This case corresponds to the left-hand configuration in Fig. 8.
- **IP/FWD:** This is a further optimized version of FWD. The network level packets are modified directly and forwarded. As a consequence, this configuration does not support reassembly of IP packets. This is the case corresponds to the right-hand configuration in Fig. 8.
- **IP Router:** This is an IP router. It also modifies network packets directly in the same way as IP/FWD, but it does not update TCP headers. It is included to show the lowest possible overhead for an intermediate host in Scout.

To compare the Scout performance with a more general-purpose operating system—so as to demonstrate that the Scout numbers are in-line with a more conventional system—we also measured the performance of a firewall and IP routing on Linux. We compiled the Linux kernel to optimize for IP routing. We consider three configurations:

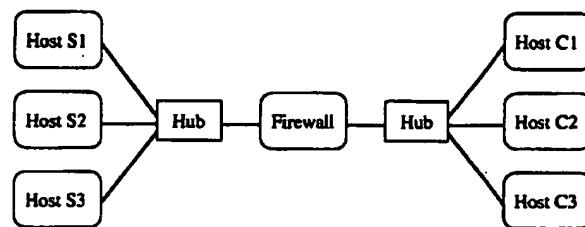


Fig. 9. Test setup.

- **TIS Firewall:** The TIS firewall toolkit offers full filter functionality [13]. We have configured it to use a null filter (plug-gw).
- **Filtering IP Router:** The in-kernel Linux IP forwarding has support for filtering on IP addresses, protocol numbers and port numbers. This is the closest thing in Linux to the IP/FWD case in Scout, except Linux neither permits starting with a proxy and later dynamically switching to the spliced connection, nor updating TCP headers.
- **IP Router:** This is the basic in-kernel Linux IP forwarding with no filtering. This shows the lowest possible overhead of the Linux configuration.

Note that while there is a myth that Linux networking performance is not very good, we have not found this to be the case with recent releases. For example, the Linux IP forwarding numbers given below are better than comparable numbers reported on BSD Unix [11]. In any case, we use the Linux numbers to calibrate the baseline case; the important results are in the incremental costs of each mechanism layered on top of this baseline.

Finally, we measure the performance of two machines connected back-to-back to evaluate the overhead of injecting a third host on the network path.

All hosts used in our experiment are 200 MHz PentiumPro workstations with 256 kB cache, 128 MB ram, and Digital Fast EtherWORKS PCI 10/100 (DE500) 32-bit PCI 10/100 Mb/s adapters. The Linux version used was 2.0.30. The physical configuration of our test setup is shown in Fig. 9. To saturate the network during throughput tests, we connected three hosts on each side of the firewall. All tests are performed between a server (hosts S1 to S3) and a client (hosts C1 to C3). In the back-to-back case, the setup was modified by connecting the two hubs to each other. All servers and clients were running Scout, as the lower complexity of Scout resulted in less variation in the measurements than Linux.

TABLE I
NON-PROCESSING RELATED OVERHEAD REMOVED FROM LATENCY MEASUREMENTS

Latency		1-byte TCP Segment	1460-byte TCP segment
Back-to-Back		77.9 μ secs	243.2 μ secs
Network Interfaces	Transmission	5.2 μ secs	121.1 μ secs
	Other	9.8 μ secs	11.7 μ secs
Total		92.9 μ secs	376.0 μ secs

TABLE II
FIREWALL AND ROUTER PROCESSING PER TCP SEGMENT

Configuration		1-byte TCP segments		1460-byte TCP segments	
		Processing time (μ secs)	Speedup	Processing time (μ secs)	Speedup
Scout	2-path	68.5	—	101.1	—
	1-path	66.1	1.04	98.6	1.03
	FWD	39.0	1.76	39.5	2.56
	IP/FWD	24.0	2.85	24.0	4.21
	IP router	22.4	3.06	22.4	4.51
Linux	TIS Firewall	83.9	—	113.0	—
	Filtering IP router	27.5	3.05	29.0	3.90
	IP router	25.5	3.29	25.4	4.45

B. Results

For all configurations, we measure the per-packet processing time for small (1-byte) and large (1460-byte) segments, and the aggregate throughput achieved with multiple connections. For Scout, we also measure the time it takes to switch from unoptimized to optimized.

1) *Processing Overhead*: To measure the per-packet processing overhead, we measured the packet round-trip times for 10 000 packets, and subtracted the back-to-back latency and network interface latency. The subtracted components are summarized in Table I. The network interface latency was obtained by measuring the processing time of a packet in the IP router configuration—that is, the time from when the packet is removed from the network interface by the interrupt handler to the time it inserted into the transmit queue of the other network interface—and subtracting this time from the total latency added by the router.

Table II summarizes the processing of a single packet in the firewalls and routers for both Scout and Linux. The 1-byte numbers reveal that connection splicing achieves a considerable speedup. Most notably, the IP/FWD case is almost a factor of three faster than application-level forwarding. In terms of packets-per-second that can be processed by the firewall, this is an increase from 14 600 to 41 600. For large packets, the speedup is even greater—a factor of four. Eliminating the extra message copy and the checksum calculations required when transferring the message from one TCP connection to another accounts for the speedup.

Also note that in both the small and large message cases, the performance of the spliced connection is very close to the performance of the IP router configuration; the TCP header transformations amount to an extra 1.6 μ s of processing. This suggests that any improvement made to IP router performance will be propagated to TCP forwarding. For example, we have found that the use of polling instead of interrupts, and the addition of a

highly optimized classification algorithm, improves IP routing performance (and hence TCP forwarding) by a factor of four [3]. On a similar note, it would be interesting to wed connection splicing with hardware supported tag switching.

Comparing the Scout and the Linux numbers, we see that the 2-path case in Scout is slightly faster than the TIS firewall on Linux. IP router performance is approximately the same for the two systems. This indicates that other types of operating systems would also benefit from connection splicing. In a Linux implementation, the IP/FWD should perform close to that of the filtering IP forwarding—the updating of the TCP and IP headers would make it slightly slower. Keep in mind, however, that simple IP filtering does not permit a proxy that can sometimes operate in control mode.

2) *Aggregate Throughput*: The sustained throughput of a TCP forwarder is also a measure of its performance. The expectation is that the improved processing overhead of the optimized forwarders should allow them to support more concurrent TCP connections.

We measured the aggregate throughput of one, two, and three concurrent TCP connections over each configuration. Each TCP connection is between a client and a server from our test setup, such that each host supports only one TCP connection. The data unit transmitted by the client process was 1460 bytes. The aggregate throughput was obtained by adding the average throughput over the last 10 s of the individual connections. This was done when the throughput had reached a stable state. Not surprisingly, these measurements turned out to be bounded by the bandwidth of the 100 Mbit Ethernet, i.e., regardless of the number of TCP connections the aggregate throughput was close to 10 Mbyte/s.

The more interesting question is how TCP forwarding behaves in the limit, that is, what bandwidth it can sustain. We can derive these numbers from the per-packet processing times presented in the previous section. For the 2-path and the IP/FWD configurations, we calculated the maximum throughput for different TCP acknowledgment patterns—either an acknowledgment

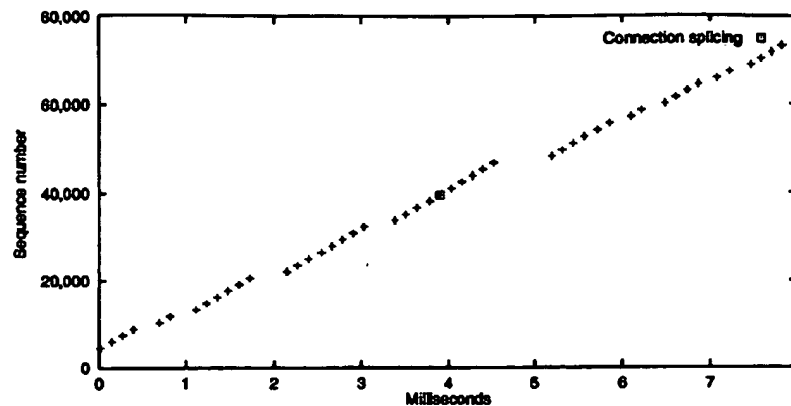


Fig. 10. TCP sequence number trace showing the effects of the Scout implementation of splicing.

TABLE III
ESTIMATED MAXIMUM THROUGHPUT OF FIREWALL IN Mbyte/s

Configuration		Message to acknowledgement ratio		
		3:1	2:1	1:1
Scout	2-path	11.7	10.8	8.6
	IP/FWD	45.6	40.6	30.4

ment is sent for every third, second, or single segment. For example, if an acknowledgment is sent for every third segment, the processing requirements for the data in the three segments would be three times the processing of a 1460-byte segment, plus the processing of a single empty segment. In our case, we have approximated the empty segment with a 1-byte segment.

The results are shown in Table III. The 2-path TCP forwarder in our measurements is operating at almost maximum bandwidth of a 100 Mbit Ethernet, whereas the IP/FWD configuration is capable of supporting up to four times the bandwidth, corresponding to two full duplex OC-3 ATM connections.

3) *Cost of Splicing*: The next question is how long it takes to splice a forwarding path. Our analysis has two parts. First, we establish the base processing overhead of splicing two TCP connections together. Second, we examine the end-to-end behavior of a TCP connection sending at maximum speed when the splicing is done.

To get the basic cost, we measured the time taken to splice two idle TCP connections on a local Ethernet.³ In this case, the measurements are free of any processing that might occur due to the draining of TCP buffers. In the test we continuously opened a TCP connection, waited 15 s and closed it again. The null proxy in the firewall optimizes the path 10 s after it is established. The numbers are the average time over 1000 such optimizations. Optimizing from TCP forwarding to FWD takes 25 μ s on average. Adding the IP/FWD forwarding takes 94 μ s on average. The higher cost of switching to IP/FWD is due to the fact that Scout requires a new path creation, whereas the FWD optimization is applied to the same path by doing code substitution.

As we concurrently forward new TCP segments and empty the buffers of the old segments, the cost of performing the optimization should be small even during high load. The more

important question is whether or not the switch affects TCP's flow or congestion control algorithms. To see the effects of the switchover on a busy TCP connection, we performed the optimization 15 s into a throughput test. By tracing the sequence numbers of segments received at the server, we were unable to see any negative effects (Fig. 10).

The expected result obviously changes depending on the RTT and bandwidth of the networks involved, especially if a lot of data have to be drained over a slow link the performance will degrade due to time-outs and out of order delivery. For example, multiple out of order delivered segments might trigger the congestion window to be halved. Since the amount of data buffered by the proxy can easily be controlled by the advertised window, it is important to only buffer enough data to deal with bursts. This will minimize the impact of the splicing operation in all scenarios.

As moving to FWD forwarding reduces the processing by an average of 27.1 μ s for small messages and 59.1 μ s for large messages, it is always a good idea to switch to the FWD optimization, independent of how much data will flow over the spliced connection. Moving from FWD to IP/FWD reduces the processing by an extra 15 μ s per packet, and thus, it will take six subsequent packets to make this optimization worthwhile.

4) *Cost of Unsplicing*: The last question is how much it costs to unsplice a connection. Unsplicing has four costs associated with it. The first cost is that, in addition to fixing up the TCP header during spliced operation, FWD also has to keep track of the sequence numbers, acknowledgment numbers, and advertised windows of the spliced TCP connection. This requires some additional state and copy operations during spliced operation.

The second cost is that FWD has to determine when to unsplice. The cost of this operation depends on the application in question. It can range from simply monitoring if data flow in a certain direction by comparing two TCP sequence numbers, to mirroring an application-level state machine on the forwarder.

The third cost is the cycles required to initiate the two independent TCP state machines. The overhead of this operation is less than generating the two TCP state machines during TCP connection establishment in the first place, since the demultiplexing part of the TCP state machine is still active.

³Using a WAN instead of a local Ethernet does not alter the results of this experiment, since all operations are local and no data have to be drained.

The last cost is the impact on end-to-end throughput. Since our implementation triggers slow start, the impact on the throughput would be quite significant for a high RTT and high bandwidth environment. This cost will likely dominate the cycle overhead of unsplicing a connection.

C. Buffer Requirements

Buffer size is an issue for large-scale TCP forwarders. First, just having enough memory to accommodate thousands of TCP connections can be a problem, as each connection can easily require up to 256 kB of buffering—two send buffers and two receive buffers of 64 kB each. This translates to buffer requirements of 256 Mbyte per 1000 TCP connections. As the use of persistent TCP connections is becoming more widespread, thousands of connections per TCP forwarder would not be uncommon. Splicing TCP connections together reduces the memory requirements of a TCP forwarder, since the forwarder is operating like an IP router and does not buffer segments. The only memory required for a spliced connection in addition to the memory required for a standard IP router is the FWD state used to fix up the IP addresses, port numbers, sequence numbers and checksum. This state can be stored in less than 36 bytes per connection—more than three orders of magnitude less memory than required for a typical TCP connection.

Dynamic buffer allocation is another solution to this problem, but it requires processing to determine how much buffer space to provide each connection. In this scenario, the TCP connections used for large data transfers are the most important. These TCP connections are the most likely candidates for splicing, thereby removing the buffer requirements altogether. In other words, splicing can also make the administration of a TCP forwarder easier.

VI. RELATED WORK

The idea of TCP splicing was developed independently by researchers at IBM [11], and its utility shown in supporting mobility [10]. Their work was done in the context of the Unix kernel, and so involves extensions to the socket interface. A more fundamental difference, however, is that the IBM approach is more restrictive than the one described in this paper. First, it supports splicing only at connection-setup time. Second, it allows only certain interactions among the client, proxy, and server. In particular:

- before the connection is spliced, only the client and the proxy can exchange data; the server is not allowed to send or receive data before the splice is complete;
- the proxy waits for an ACK of all data it has sent the client before engaging the splice;
- once the splice is in place, the client is allowed to send data to the server. It is the arrival of these data at the server that notify the server that the splice is complete; the server is not allowed to send until this time.

This interaction is enforced by the SOCKS library package that must be linked with both the client and the proxy [9].

In contrast, our approach allows the splicing optimization to be transparently engaged at any point in the lifetime of the

two TCP connections, including after the client and server have exchanged data. This is accomplished by having the proxy simultaneously process buffered data and forward newly arriving data, as described in Section III-C. The important consequence of this difference is that our approach allows the proxy to arbitrarily filter the data passed between the client and server before it initiates the splice and removes itself from the path. This means, for example, that a proxy is able to parse a URL in an HTTP stream. The IBM approach does not support such general filtering.

More broadly, TCP forwarders are used to separate the TCP connection on a wireless link from that of a wired network [2]. This increases performance as the characteristics of the two types of networks are very different. As a mobile host moves around, it might sometimes connect directly to a wired network, in which case the TCP forwarder becomes superfluous and can be removed. This is done in the TACO system [8], where mobile hosts can—depending on what is required from their current type of network attachment—switch between having a TCP forwarder and not, without destroying their TCP connections. The system differs from the one presented in this paper in two ways: it does not support filtering, and it uses interleaved connection establishment. This allows the TCP forwarder to be removed completely from the network path in the optimized case as no translation is necessary, but it at the same time limits the applicability of the solution. The lack of filtering makes it unsuitable for more advanced proxies such as firewalls.

Another research topic related to this paper is that of efficiently classifying packets [1], [7]. Of particular note are new algorithms to do fast routing table lookups based on variable length IP address prefixes [4], [14]. It is easy to imagine such techniques being extended to support fast IP filtering. Such an advance would be complementary to connection splicing, which can also exploit improved algorithms to determine to which path a particular packet belongs. Connection splicing is more general than IP filtering, however, since the proxy permits complex control operations.

VII. CONCLUDING REMARKS

This paper describes connection splicing, which can be applied to TCP forwarders to improve their performance. A performance study shows that an optimized TCP forwarder requires between one-half and one-quarter of the processing requirements of an unoptimized forwarder. The cost of the optimization varies according to how fast the buffers at the TCP forwarder can be emptied, but in most cases the cost is recovered within one to six packets. Furthermore, the optimization reduces the memory requirements of a TCP forwarder. The optimizations have been implemented in the Scout operating system, and it should be possible to get equivalent performance improvements in other systems.

In the future we would like to investigate when and how splicing should be applied in the emerging fields of content distribution and application-level routing. Of particular interest is the impact of splicing on persistent and SSL-secured HTTP connections.

ACKNOWLEDGMENT

The authors would like to thank the other members of the Network Systems Group at both the University of Arizona and Princeton University, and in particular, G. Tong, who implemented unsplicing. They would also like to thank the anonymous reviewers who provided helpful feedback on the manuscript.

REFERENCES

- [1] M. L. Bailey, B. Gopal, M. A. Pagels, L. L. Peterson, and P. Sarkar, "PathFinder: A pattern-based packet classifier," in *Proc. 1st Symp. Operating Systems Design and Implementation*, Monterey, CA, 1994, pp. 115–123.
- [2] A. Bakre and B. R. Badrinath, "Implementation and performance evaluation of indirect TCP," *IEEE Trans. Comput.*, vol. 46, no. 3, Mar. 1997.
- [3] A. Bavier, S. Karlin, L. Peterson, and X. Qie, "Scheduling computations on programmable routers," Princeton Univ., Princeton, NJ, Tech. Rep. 615-00, Feb. 2000.
- [4] A. Brodnik, S. Carlsson, M. Degermark, and S. Pink, "Small forwarding tables for fast routing lookups," in *Proc. ACM SIGCOMM'97 Symp.*, Cannes, France, Sep. 1997, pp. 3–14.
- [5] C. Brooks, M. Mazer, S. Meeks, and J. Miller, "Application-specific proxy servers as HTTP stream transducers," in *Electronic Proc. 4th Int. World Wide Web Conf., "The Web Revolution"*, Boston, MA, Dec. 1995.
- [6] M. Chatel, "RFC 1919: Classical versus transparent IP proxies," Mar. 1996.
- [7] D. Engler and M. F. Kaashoek, "DPF: Fast, flexible message demultiplexing using dynamic code generation," in *Proc. ACM SIGCOMM'96 Symp.*, Stanford, CA, Aug. 1996, pp. 53–59.
- [8] J. S. Hansen, T. Reich, B. Andersen, and E. Jul, "Dynamic adaptation of network connections in mobile environments," *IEEE Internet Computing*, vol. 2, no. 1, Jan/Feb. 1998.
- [9] M. Leech *et al.*, "SOCKS protocol version 5," RFC 1928, Mar. 1996.
- [10] D. Maltz and P. Bhagwat, "MSOCKS: An architecture for transport layer mobility," in *Proc. IEEE INFOCOM*, Apr. 1998, [ftp://ftp.monarch.cs.cmu.edu/pub/dmaltz/msocks-infocom98.ps.gz](http://ftp.monarch.cs.cmu.edu/pub/dmaltz/msocks-infocom98.ps.gz), pp. 1037–1045.
- [11] —, "TCP splicing for application layer proxy performance," IBM, [ftp://ftp.cs.cmu.edu/user/dmaltz/Doc/splice-perf-tr.ps](http://ftp.cs.cmu.edu/user/dmaltz/Doc/splice-perf-tr.ps), Mar. 1998.
- [12] D. Mosberger and L. Peterson, "Making paths explicit in the Scout operating system," in *Proc. 2nd Symp. Operating Systems Design and Implementation*, Oct. 1996, pp. 153–168.
- [13] M. K. Ranum and F. M. Avolio, "A toolkit and methods for Internet firewalls," in *Proc. Summer 1994 USENIX Conf.*, Berkeley, CA, USA, June 1994, pp. 37–44.
- [14] M. Waldvogel, G. Varghese, J. Turner, and B. Plattner, "Scalable high speed IP routing lookups," in *Proc. ACM SIGCOMM'97 Symp.*, Cannes, France, Sep. 1997, pp. 25–38.

Oliver Spatscheck was born in Germany. He received the M.S. and Ph.D. degrees in computer science from the University of Arizona, Tucson, in 1996 and 1999, respectively.

He is a Senior Technical Staff Member at AT&T Labs—Research, Florham Park, NJ, where he works on differential services, denial of service prevention, and intelligent content distribution.

Jørgen S. Hansen (S'96) received the M.S. degree in computer science in 1996 from the University of Copenhagen, Denmark, where he is currently pursuing the Ph.D. degree, also in computer science.

He visited the University of Arizona, Tucson, for seven months in 1997–1998. His research interests include operating system support for high-speed networking, distributed systems, and mobile computing.

Mr. Hansen is a student member of ACM.

John H. Hartman (M'95) received the Sc.B. degree in computer science from Brown University, Providence, RI, in 1987, and the M.S. and Ph.D. degrees in computer science from the University of California, Berkeley, in 1990 and 1994 respectively.

He has been an Assistant Professor in the Department of Computer Science, University of Arizona, Tucson, since 1995. His research interests include distributed systems, operating systems, and file systems.

Dr. Hartman is a member of ACM.

Larry L. Peterson (SM'95) received the B.S. degree in computer science from Kearney State College, Kearney, NE, in 1979, and the M.S. and Ph.D. degrees in computer science from Purdue University, West Lafayette, IN, in 1982 and 1985 respectively.

He is a Professor of computer science at Princeton University, Princeton, NJ. Prior to joining Princeton University, he was the Head of the Computer Science Department, University of Arizona, Tucson. His research focuses on end-to-end issues related to computer networks. He has been involved in the design and implementation of x-kernel and Scout operating systems. He is a coauthor of the textbook *Computer Networks: A Systems Approach* (San Mateo, CA, Morgan Kaufman, 2000).

Dr. Peterson is the Editor-in-Chief of the *ACM Transactions on Computer Systems*. He has served on the editorial boards for *IEEE/ACM TRANSACTIONS ON NETWORKING* and the *IEEE JOURNAL ON SELECTED AREAS IN COMMUNICATION*, and on program committees for SOSP, SIGCOMM, OSDI, and ASPLOS. He is also a member of the Internet's End-to-End research group, and a fellow of ACM.

Kernel Mechanisms for Service Differentiation in Overloaded Web Servers

Thiemo Voigt*

Swedish Institute of Computer Science

thiemo@sics.se

Renu Tewari

IBM T.J. Watson Research Center

tewarir@us.ibm.com

Douglas Freimuth

IBM T.J. Watson Research Center

dmfreim@us.ibm.com

Ashish Mehra

iScale Networks

ashish@iscale.net

Abstract

The increasing number of Internet users and innovative new services such as e-commerce are placing new demands on Web servers. It is becoming essential for Web servers to provide performance isolation, have fast recovery times, and provide continuous service during overload at least to preferred customers. In this paper, we present the design and implementation of three kernel-based mechanisms that protect Web servers against overload by providing admission control and service differentiation based on connection and application level information. Our basic admission control mechanism, *TCP SYN policing*, limits the acceptance rate of new requests based on the connection attributes. The second mechanism, *prioritized listen queue*, supports different service classes by reordering the listen queue based on the priorities of the incoming connections. Third, we present *HTTP header-based connection control* that uses application-level information such as URLs and cookies to set priorities and rate control policies.

We have implemented these mechanisms in AIX 5.0. Through numerous experiments we demonstrate their effectiveness in achieving the desired degree of service differentiation during overload. We also show that the kernel mechanisms are more efficient and scalable than application level controls implemented in the Web server.

*This work was partially funded by the national Swedish Real-time Systems Research Initiative (ARTES). This work was done when the author was visiting the IBM T.J. Watson Research Center.

1 Introduction

Application service providers and Web hosting services that co-host multiple customer sites on the same server cluster or large SMP are becoming increasingly common in the current Internet infrastructure. The increasing growth of e-commerce on the web means that any server down time that affects the clients being serviced will result in a corresponding loss of revenue. Additionally, the unpredictability of flash crowds can overwhelm a hosting server and bring down multiple customer sites simultaneously, affecting the performance of a large number of clients. It becomes essential, therefore, for hosting services to provide performance isolation and continuous operation under overload conditions.

Each of the co-hosted customers sites or applications may have different quality-of-service (QoS) goals based on the price of the service and the application requirements. Furthermore, each customer site may require different services during overload based on the client's identity (preferred gold client) and the application or content they access (e.g., a client with a buy order vs. a browsing request). A simple threshold based request discard policy (e.g., a TCP SYN drop mode in commercial switches/routers discards the incoming, oldest or any random connection [1]) to delay or control overload is not adequate as it does not distinguish between the individual

QoS requirements. For example, it would be desirable that requests of non-preferred customer sites be discarded first. Such QoS specifications are typically negotiated in a service level agreement (SLA) between the hosting service provider and its customers. Based on this governing SLA, the hosting service providers need to support service differentiation based on client attributes (IP address, session id, port etc.), server attributes (IP address, type), and application information (URL accessed, CGI request, cookies etc.).

In this paper, we present the design and implementation of kernel mechanisms in the network subsystem that provide admission control and service differentiation during overload based on the customer site, the client, and the application layer information.

One of the underlying principles of our design was that it should enable "early discard", i.e., if a connection is to be discarded it should be done as early as possible, before it has consumed a lot of system resources [2]. Since a web server's workload is generated by incoming network connections we place our control mechanisms in the network subsystem of the server OS at different stages of the protocol stack processing. To balance the need for early discard with that of an informed discard, where the decision is made with full knowledge of the content being accessed, we provide mechanisms that enable content-based admission control.

Our second principle was to introduce minimal changes to the core networking subsystem in commercial operating systems that typically implement a BSD-style stack. There have been prior research efforts that modify the architecture of the networking stack to enable stable overload behavior [3]. Other researchers have developed new operating system architectures to protect against overload and denial of service attacks [4]. Some "virtual server" implementations try to sandbox all resources (CPU, memory, network bandwidth) according to administrative policies and enable complete performance isolation [5]. Our aim in this design, however, was not to build a new networking architecture but to introduce simple controls in the existing architecture that could be just as effective.

The third principle was to implement mechanisms that can be deployed both on the server as well as outside the server in layer 4 or 7 switches that perform load balancing and content based routing

for a server farm or large cluster [6]. Such switches have some form of overload protection mechanisms that typically consists of dropping a new connection packet (or some random new connection packet) when a load threshold is exceeded. For content-based routing the layer 7 switch functionality consists of terminating the incoming TCP connection to determine the destination server based on the content being accessed, creating a new connection to the server in the cluster, and splicing the two connections together [7]. Such a switch has access to the application headers along with the IP and TCP headers. The mechanisms we built in the network subsystem can easily be moved to the front-end switch to provide service differentiation based on the client attributes or the content being accessed.

There have been proposals to modify the process scheduling policies in the OS to enable preferred web requests to execute as higher priority processes [8]. These mechanisms, however, can only change the relative performance of higher priority requests; they do not limit the requests accepted. Since the hardware device interrupt on a packet receive and the software interrupt for packet protocol processing can preempt any of the other user processes [3] such scheduling policies cannot prevent or delay overload. Secondly, the incoming requests already have numerous system resources consumed before any scheduling policy comes into effect. Such priority scheduling schemes can co-exist with our controls in the network subsystem.

An alternate approach is to enable the applications to provide their individual admission control mechanisms. Although this achieves application level control it requires modifications to existing legacy applications or specialized wrappers. Application controls are useful in differentiating between different clients of an application but are less useful in preventing or delaying overload across customer sites. More importantly, various server resources have already been allocated to a request before the application control comes into effect, violating the early discard policy. However, the kernel mechanisms can easily work in conjunction with application specific controls.

Since most web servers receive requests over HTTP/TCP connections, our controls are located in three different stages in the lifetime of a TCP connection.

- The first control mechanism, *TCP SYN policing*, is located at the start of protocol stack processing of the first SYN packet of a new connection and limits acceptance of a new TCP SYN packet based on compliance with a token bucket based policer.
- The next control, *prioritized listen queue*, is located at the end of a TCP 3-way handshake, i.e., when the connection is accepted and supports different priority levels among accepted connections.
- Third, *HTTP header-based connection control*, is located after the HTTP header is received (which could be after multiple data packets) and enables admission control and priority values to be based on application-layer information contained in the header e.g., URLs, cookies etc.

We have implemented these controls in the AIX 5.0 kernel as a loadable module using the framework of an existing QoS-architecture [9]. The existing QoS architecture on AIX supports policy-based outbound bandwidth management [10]. These techniques are easily portable to any OS running a BSD style network stack¹.

We present experimental results to demonstrate that these mechanisms effectively provide selective connection discard and service differentiation in an overloaded server. We also compare against application layer controls that we added in the Apache 1.3.12 server and show that the kernel controls are much more efficient and scalable.

The remainder of this paper is organized as follows: In Section 2 we give a brief overview on input packet processing. Our architecture and the kernel mechanisms are presented in Section 3. In Section 4 we present and discuss experimental results. We compare the performance of kernel based mechanisms and application level controls in Section 5. We describe related work in Section 6 and finally, the conclusions and future work in Section 7.

2 Input Packet Processing: Background

In this section we briefly describe the protocol processing steps executed when a new connection re-

¹A port to Linux is underway.

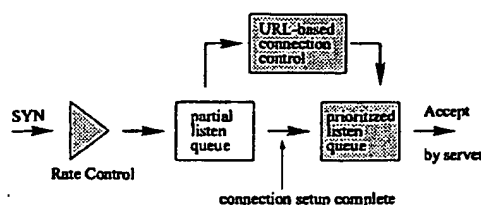


Figure 1: Proposed kernel mechanisms.

quest is processed by a web server. When the device interface receives a packet it triggers a hardware interrupt that is serviced by the corresponding device driver [11]. The device driver copies the received packet into an mbuf and de-multiplexes it to determine the queue to insert the packet. For example, an IP packet is added to the input queue, `ipintrq`. The device driver then triggers the IP software interrupt. The IP input routine dequeues the packet from the IP input queue and does the next layer de-multiplexing to invoke the transport layer input routine. For example, for a TCP packet this will result in a call to a `tcp_input` routine for further processing. The call to the transport layer input routine happens within the realm of the IP input call, i.e., there is no queuing between the IP and TCP layer. The TCP input processing verifies the packet and locates the protocol control block (PCB). If the incoming packet is a SYN request for a listen socket, a new data socket is created and placed in the partial listen queue and an ACK is sent back to the client. When the ACK for the SYN-ACK is received the TCP 3-way handshake is complete, the connection moves to an established state and the data socket is moved to the listen queue. The sleeping process, e.g., the web server, waiting on the accept call is woken up. The connection is ready to receive data.

3 Architecture Design

The network subsystem architecture adds three control mechanisms that are placed at the different stages of a TCP connection's life time. Figure 1 shows the various phases in the connection setup and the corresponding control mechanisms: (i) when a SYN packet is processed it triggers the SYN rate control and selective drop (ii) when the 3-way handshake is completed the prioritized listen queue selectively changes the ordering of accepted connections in the listen queue (iii) when the HTTP header is received the HTTP header controls decide on dropping or re-prioritizing the requests based on application

layer information. Each of these mechanisms can be activated at varying degrees of overload where the earliest and simplest control is triggered at the highest load level.

3.1 SYN Policer

TCP SYN policing controls the rate and burst at which new connections are accepted. Arriving TCP SYN packets are policed using a token bucket profile defined by the pair $\langle rate, burst \rangle$, where *rate* is the average number of new requests admitted per second and *burst* is the maximum number of concurrent new requests. Incoming connections are aggregated using specified filter rules that are based on the connection end points (source and destination addresses and ports as shown in Table 2). On arrival at the server, the SYN packet is classified using the IP/TCP header information to determine the matching rule. A compliance check is performed against the token bucket profile of the rule. If compliant, a new data socket is created and inserted in the partial listen queue otherwise the SYN packet is silently discarded.

When the SYN packet is silently dropped, the requesting client will time-out waiting for a SYN ACK and retry again with an exponentially increasing time-out value². An alternate option, which we do not consider, is to send a TCP RST to reset the connection indicating an abort from the server. This approach, however, incurs unnecessary extra overhead. Secondly, some clients send a new SYN immediately after a TCP RST is received instead of aborting the connection. Note that we drop non-compliant SYNs even *before* a socket is created for the new connection thereby investing only a small amount of overhead on requests that are dropped.

To provide service differentiation, connection requests are aggregated based on filters and each aggregate has a separate token bucket profile. Filtering based on client IP addresses is useful since a few domains account for a significant portion of a web server's requests [12]. The rate and burst values are enforced only when overload is detected and can be dynamically controlled by an adaptation agent, the details of which are beyond the scope of this paper.

²The timeout values are typically set to 6, 24, 48, up to 75 seconds.

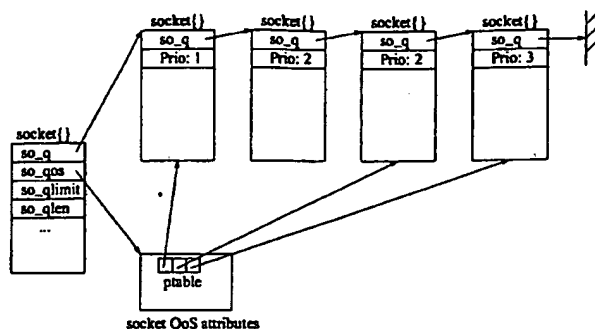


Figure 2: Implementation of the prioritized listen queue

3.2 Prioritized Listen Queue

The prioritized listen queue reorders the listen queue of a server process based on pre-defined connection priorities such that the highest priority connection is located at the head of the queue. The priorities are associated with filters (see Table 2) and connections are classified into different *priority classes*. When a TCP connection is established, it is moved from the partial listen queue to the listen queue. We insert the socket at the position corresponding to its priority in the listen queue. Since the server process always removes the head of the listen queue when calling *accept*, this approach provides better service, i.e. lower delay and higher throughput, to connections with higher priority.

Figure 2 shows the implementation of a prioritized listen queue. A special data structure used for maintaining socket QoS attributes stores an array of *priority pointers*. Each priority pointer points to the *last* socket of the corresponding priority class. This allows efficient socket insertion — a new socket is always inserted behind the one pointed to by the corresponding priority pointer.

3.3 HTTP Header-based Controls

The SYN policer and prioritized listen queue have limited knowledge about the type and nature of a connection request, since they are based on the information available in the TCP and IP headers. For web servers with the majority of the traffic being HTTP over TCP, a more informed control is possible by examining the HTTP headers. For example,

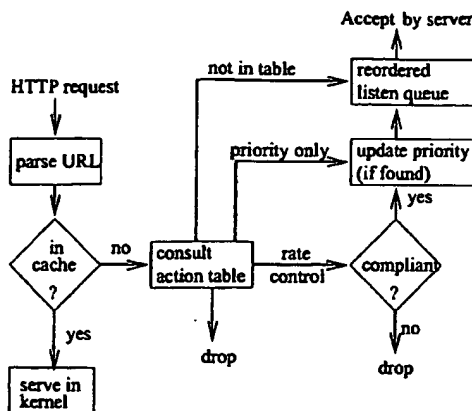


Figure 3: The HTTP header-based connection control mechanism.

Table 1: URL action table

URL	ACTION
noaccess	<drop>
/shop.html	<priority=1>
/index.html	<rate=15 conn./sec, burst=5 conn.>, <priority=1>
/cgi-bin/*	<rate=10, burst=2>

a majority of the load is caused by a few CGI requests and most of the bytes transferred belong to a small set of large files. This suggests that targeting specific URLs, types of URLs, or cookie information for service differentiation can have a wide impact during overload.

Our third mechanism, *HTTP header-based connection control*, enables content-based connection control by examining application layer information in the HTTP header, such as the URL name or type (e.g., CGI requests) and other application-specific information available in cookies. The control is applied in the form of rate policing and priorities based on URL names and types and cookie attributes.

This mechanism involves parsing the HTTP header in the kernel and waking the sleeping web server process only after a decision to service the connection is made. If a connection is discarded, a TCP RST is sent to the client and the socket receive buffer contents are flushed.

For URL parsing, our implementation relies upon Advanced Fast Path Architecture (AFPA) [13], an

Table 2: Example Network-level Policies

(dst IP, dst port, src IP, src port)	(r,b)	priority
(*, 80, *, *)	(300,5)	3
(*, 80, 10.1.1.1, *)	(100,5)	2
(12.1.1.1, 80, *, *)	(10,1)	*

in-kernel web cache on AIX. For Linux, an in-kernel web engine called KHTTPD is available [14]. As opposed to the normal operation, where the sleeping process is woken up after a connection is established, AFPA responds to cached HTTP requests directly without waking up the server process. With AFPA, a connection is *not* moved out of the partial listen queue even after the 3-way handshake is over. The normal data flow of TCP continues with the data being stored in the socket receive buffer. When the HTTP header is received (that is when the AFPA parser finds two CR control characters in the data stream), AFPA checks for the object in its cache. On a cache miss, the socket is moved to the listen queue and the web server process is woken up to service the request.

The HTTP header-based connection control mechanism comes into play at this juncture, as illustrated in Figure 3, before the socket is moved out of the partial listen queue. The URL action table (Table 1) specifies three types of actions/controls for each URL or set of URLs. A drop action implies that a TCP RST is sent before discarding the connection from the partial listen queue and flushing the socket receive buffer. If a priority value is set it determines the location of the corresponding socket in the ordered listen queue. Finally, rate control specifies a token bucket profile of a <rate, burst> pair which drops out-of-profile connections similar to the SYN policer.

3.4 Filter Specification

A filter rule specifies the network-level and/or application-level attributes that define an aggregate and the parameters for the control mechanism that is associated with it. A network-level filter is a four-tuple consisting of local IP address, local port, remote IP address, and remote port; application-level filters were shown in Table 1. Table 2 lists some network-level filter examples. The first rule applies to the web server process listening at local port 80 on all network interfaces; it specifies that all con-

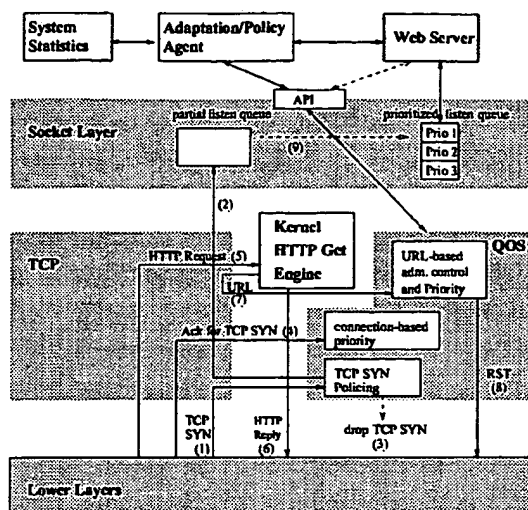


Figure 4: Enhanced protocol stack architecture.

nections to the server are rate-controlled at a rate of 300 conns/sec, a burst of 5, and a priority of 3 (the default lowest priority). The filter rules can contain range of IP addresses, wildcards, etc.

3.5 Protocol Stack Architecture

We have developed architectural enhancements for Unix-based servers to provide these mechanisms. Figure 4 shows the basic components of the enhanced protocol stack architecture, with the new capabilities utilized either by user-space agents or applications themselves. This architecture permits control over an application's inbound network traffic via policy-based traffic management [10]; an adaptation/policy agent installs policies into the kernel via a special API. The policy agent interacts with the kernel via an enhanced socket interface by sending (receiving) messages to (from) special control sockets. The policies specify filters to select the traffic to be controlled, and actions to perform on the selected traffic. The figure shows the flow of an incoming request through the various control mechanisms.

3.6 Implementation Methodology and Testbed

We have implemented the proposed kernel mechanisms in AIX 5.0, and evaluated them on the testbed

described below. As shown in Figure 4, the QoS module contains the TCP SYN policer, a priority assignment function for new connections, and the entity that performs URL-based admission control and priority assignment.

All experiments were conducted on a testbed comprising an IBM HTTP Server running on a 375 MHz RS/6000 machine with 512 MB memory, several 550 MHz Pentium III clients running Linux, and one 166 MHz Pentium Pro client running FreeBSD. The server and clients are connected via a 100 BaseT Ethernet switch. For client load generators we use Webstone 2.5 [15] and a slightly modified version of sclient [16]. Both programs measure client throughput in connections per second. The experimental workload consists of static and dynamic requests. The dynamic files are minor modifications of standard Webstone CGI files that simulate memory consumption of real-world CGIs.

The IBM HTTP Server is a modified Apache [17] 1.3.12 web server that utilizes an in-kernel HTTP get engine called the Advanced Fast Path Architecture (AFPA). We use AFPA in our architecture only to perform the URL parsing and have disabled any caching when measuring throughput results. Unless stated otherwise, we configured Apache to use a maximum of 150 server processes.

4 Experimental Results

4.1 Efficacy of SYN Policing

In this section we show how TCP SYN policing protects a preferred client against flash crowds or high request rates from other clients. In our setup, one client replays a large e-tailer's trace file representing a preferred customer. For the competing load we use five machines running Webstone, each with 50 clients. All clients request an 8 KB file, which is reasonable since a typical HTTP transfer is between 5 and 13 KB [12].

Without SYN policing, the e-tailer's client receives a low throughput of about 6 KB/sec. Using policing to lower the acceptance rate of Webstone clients, we expect the throughput for the e-tailer's client to increase. Figure 5 shows that the throughput for e-tailer's client increases from 100 KB/sec to 800 KB/sec as the acceptance rate for Webstone clients

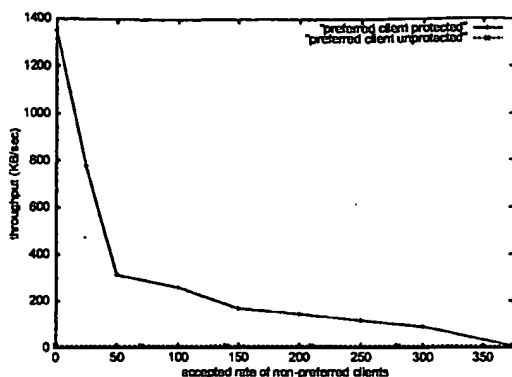


Figure 5: Throughput of the preferred e-tailer's client with and without TCP SYN policing. On the X-axis is the SYN policing rate of the non-preferred Webstone clients that are continuously generating requests. The Y-axis shows the corresponding throughput received by the e-tailer's client when there was no SYN control and when SYN control was enforced.

is lowered from 300 reqs/sec to 25 reqs/sec. The experiment demonstrates that a preferred client can be successfully protected by rate-controlling connection requests of other greedy clients.

TCP SYN policing works well when client identities and request patterns are known. In general, however, it is difficult to correctly identify a misbehaving group of clients. Moreover, as discussed below, it is hard to predict the rate control parameters that enable service differentiation for preferred clients without under-utilizing the server. For effective overload prevention the policing rate must be dynamically adapted to the resource consumption of accepted requests.

4.2 Impact of Burst Size

In the previous experiment we did not analyze the effect of the burst size on the effective throughput. The burst size is the maximum number of new connections accepted concurrently for a given aggregate. With a large burst size, greedy clients can overload the server, whereas with a small burst, clients may be rejected unnecessarily. The burst size also controls the responsiveness of rate control. There is a tradeoff, however, between responsiveness and the achieved throughput.

We next show the effect of the burst size on the

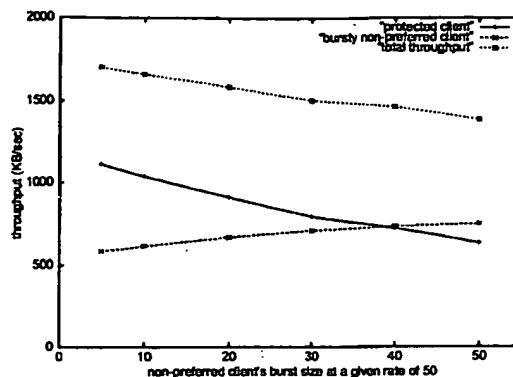


Figure 6: Impact of burst size on preferred client throughput. The burst size for policing non-preferred client is varied from 5 to 50 while the connection acceptance rate is fixed at 50 conn/sec. The plot shows the throughput achieved by the preferred and non-preferred clients along with the total throughput.

throughput of a preferred client. In our experiment, the non-preferred client is a modified sclient program that makes 50 to 80 back-to-back connection requests about twice a second, in addition to the specified request rate. Both the length of the incoming request burst and its timing are randomized. Figure 6 shows the throughput of preferred and non-preferred client with the SYN policing rate of the non-preferred client set to 50 conn/sec and the burst size varying from 5 to 50. The non-preferred sclient program requests a 16 KB dynamically generated cgi file. The preferred client is a Webstone program with 40 clients, requesting a static 8 KB file. As the burst size is increased from 5 to 50, the sclient's throughput increases from 36.6 conns/sec (585.6 KB/sec) to 47.7 conns/sec (752 KB/sec), while the throughput received by the preferred client decreases from about 140 conns/sec (1117 KB/sec) to 79 conns/sec.

Intuitively the overall throughput should have increased, however, the observed decrease in total throughput is due to the fact that we accept more CPU consuming CGI requests from sclient, thereby, incurring a higher overhead per byte transferred.

4.3 Prioritized Listen Queue: Simple Priority

With TCP SYN policing, one must limit the greedy non-preferred clients to a meaningful rate during overload. In most cases it is relatively simpler to just give the preferred clients a higher absolute pri-

ority. We demonstrate next that the prioritized listen queue provides service differentiation, especially with a large listen queue length.

In our experiments we classify clients into three priority levels. Clients belonging to a common priority level are all created by a Webstone benchmark that requests an 8 KB file. A separate Webstone instance is used for each priority level. We measure client throughput for each priority level while varying the total number of clients in each class. Each priority class uses the same number of clients.

In the first experiment, the Apache server is configured to spawn a maximum of 50 server processes. The results in Figure 7 show that when the total number of clients is small, all priority levels achieve similar throughput. With fewer clients, server processes are always free to handle incoming requests. Thus, the listen queue remains short and almost no reordering occurs. As the number of clients increases, the listen queue builds up since there are fewer Apache processes than concurrent client requests. Consequently, with re-ordering the throughput received by the high priority client increases, while that of the two lower priority clients decreases. Figure 7 shows that with more than 30 Webstone clients per class only the high-priority clients are served while the lower-priority clients receive almost no service.

Figure 8 illustrates the effect on response times observed by clients of the three priority classes. It can be seen that as the number of clients increases across all priority classes the response time for the lower priority classes increases exponentially. The response time of the high priority class, on the other hand, only increases sub-linearly. When the number of high priority requests increases, the lower priority ones are shifted back in the listen queue, thereby, increasing their response times. Also as more high priority requests get serviced by the different server processes running in parallel and competing for the CPU their response times increase.

We also observed that when the number of high priority requests was fixed and the lower priority request rate was steadily increased, the response time of the high priority requests remained unaffected.

The priority-based approach enables us to give low delay and high throughput to preferred clients independent of the requests or request patterns of

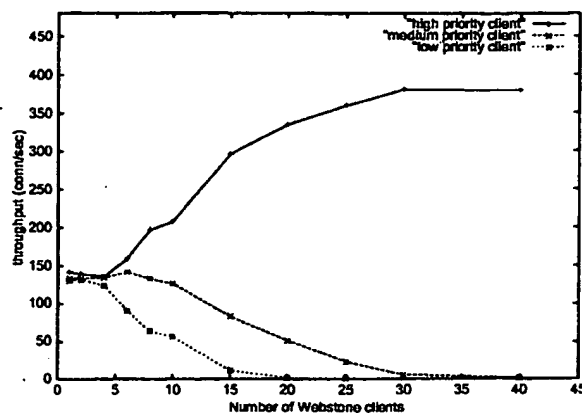


Figure 7: Throughput with the prioritized listen queue and 3 priority classes with 50 Apache processes. The number of clients in each class remains equal.

other clients. However, one may need many priority classes for different levels of service. The main drawback of a simple priority ordering is that it provides no protection against starvation of low-priority requests.

4.4 Combining Policing and Priority

To prevent starvation, low priority requests need to have some minimum number reserved slots in the listen queue so that they are not always preempted by a high priority request. However, reserving slots in the listen queue arbitrarily could cause a high priority request to find a full listen queue, which would in turn cause it to be aborted after its 3-way handshake is completed. To avoid starvation with fixed priorities, we combine the listen queue priorities with SYN policing to give preferred clients higher priority, but limiting their maximum rate and burst, thereby, implicitly reserving some slots in the queue for the lower priority requests.

Table 3 shows the results for experiments with three sets of Webstone clients with different priorities and rate control of the high priority class. The lower priority class has 30 Webstone clients while the high priority class has 150 Webstone clients spread over three different hosts. With no SYN policing of the clients in the high priority class, the two low-priority clients are completely starved. Table 3 shows that rate limiting the clients in the high priority class to 300 conn/sec prevents starvation; the medium and

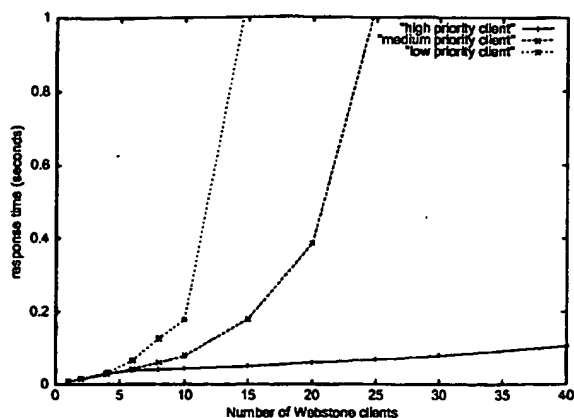


Figure 8: Response time with the prioritized listen queue and 3 priority classes with 50 Apache processes. The number of clients in each class remains equal.

Table 3: TCP SYN policing of a high-priority client to avoid starvation of other clients.

Throughput (conn/sec) of each priority class			
client priority	(rate, burst) limit of high priority		
	none	(300,300)	(200,200)
high	381	306	196
medium	0	78.6	180
low	0	4.1	13

low priority clients achieve a throughput of 78.6 and 4.1 conn/sec respectively.

4.5 HTTP Header-based Connection Control

In this section we illustrate the performance and effectiveness of admission control and service differentiation based on information in the HTTP headers i.e., URL name and type, cookie fields etc.

Rate control using URLs: In our experimental scenario the preferred client replaying the e-tailer's trace needs to be protected from overload due to a large number of high overhead CGI requests from non-preferred clients. The client issuing CGI requests is an sclient program requesting a dynamic file of length 5 KB at a very high rate. Figure 9 shows that without any protection, the preferred e-tailer's customer receives a low throughput of under 1 KB/sec. By rate-limiting the dynamic requests

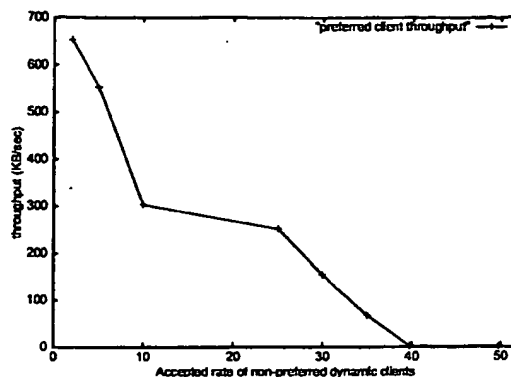


Figure 9: URL-based policing to protect preferred e-tailer's customers. The graph shows the resulting throughput of the preferred e-tailer's client as a specific high overhead CGI requests is limited to a given number of conn/sec

from 40 reqs/sec to 2 reqs/sec the throughput of the preferred e-tailer's customer improves from 1 KB/sec to 650 KB/sec. In contrast to TCP SYN policing (Figure 5), URL rate control targets a specific URL causing overload instead of a client pool.

URL priorities: In this section we present the results of priority assignments in the listen queue based on the URL name or type being requested. The clients are Webstone benchmarks requesting two different URLs, both corresponding to files of size 8 KB. There are two priority classes in the listen queue based on the two requested URLs. Figure 10 shows that the lower priority clients (accessing the low priority URL) receive lower throughput and are almost starved when the number of clients requesting the high priority URL exceeds 40. These results correspond to the results shown earlier with priorities based on the connection attributes (see Figure 7). The average total throughput, however, is slightly lower with URL-based priorities due to the additional overhead of URL parsing.

Combined URL-based rate control and priorities: To avoid starvation of requests for the low-priority URL, we rate limit the requests for the high-priority URL. In this experiment, we assign a higher priority to requests for a dynamic CGI request of size 5 KB (requested by an sclient program), and lower priority to requests for a static 8 KB file (requested by the Webstone program). Table 4 shows that starvation can be avoided by rate-limiting the high-priority URL requests.

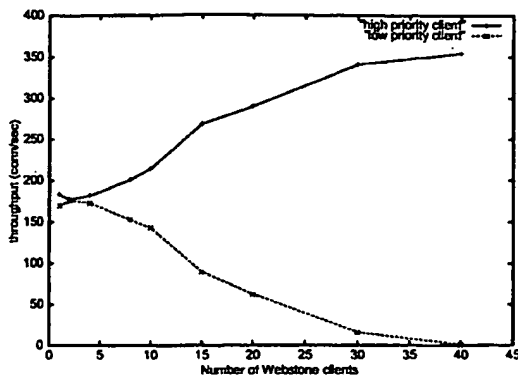


Figure 10: Throughput with 2 URL-based priorities and 50 Apache server processes. The number of clients in each class is equal

Table 4: URL-based policing of a high-priority client to avoid starvation of other clients.

Throughput (conn/sec)			
client priority	(rate, burst) limit of high priority		
	none	(30,10)	(10,10)
high	61.7	29.0	10.1
low	0	10.2	117

4.5.1 Overload Protection from High Overhead Requests

So far we have used the URL-based controls for providing service differentiation based on URL names and types. In the next experiment, we investigate if URL-based connection control can be used to protect a web server from overload by a targeted control of high overhead requests (e.g., CGI requests that require large computation or database access).

We use the sclient load generator to request a given high overhead URL and control the request rate, steadily increasing it and measuring the throughput. Figure 11 shows the client's throughput with varying request rates for a dynamic CGI request that generates a file size of 29 KB. The throughput increases linearly with the request rate up to a critical point of about 63 connections/sec. For any further increase in the request rate the throughput falls exponentially and later plateaus to around 40 connections/sec. To understand this behavior we used vmstat to capture the paging statistics. Since the dynamic requests are memory-intensive, the available free memory rapidly declines. For some combinations of the request rate

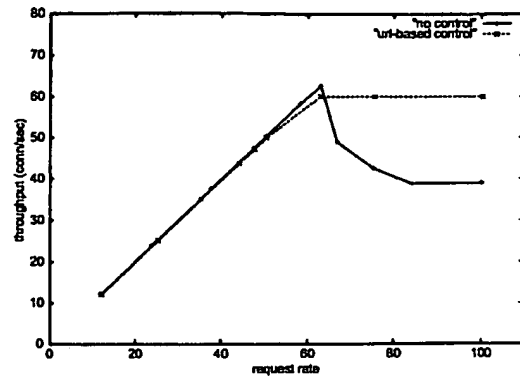


Figure 11: Overload protection from high overhead requests using URL-based connection control. The graph shows the throughput of web server with no controls servicing CPU intensive CGI requests and the corresponding throughput when the CGI requests are limited to 60 reqs/sec.

and the number of active processes, the available free memory falls to zero. Eventually the system starts thrashing as the CPU spends most of the time waiting for pending local disk I/O. In the above experiment with 150 server processes and a request rate of 63 reqs/sec the wait time starts increasing as indicated by the wait field of the output from vmstat.

To prevent overload we use URL-based connection control to limit the number of accepted dynamic CGI requests to a rate of 60 reqs/sec and a burst of 10. The dashed line in Figure 11 shows that with URL-based control the throughput stabilizes to 60 reqs/sec and the server never thrashes. In the above experiment, the URL-based connection control can handle request rates of up to 150 requests per second. However, for request rates beyond that thrashing starts as the kernel overhead of setting up connections, parsing the URL and sending the RSTs, becomes substantial.

To further delay the onset of thrashing we augment the URL-based control with the TCP SYN policer. For every TCP RST that is sent we drop any subsequent SYN request from that same client for a specified time interval. The time interval selected is the timeout value used for a lost SYN.

Table 5: Performance of AFPA and matching a URL to a rule for a 8 KB file with different URL lengths.

Throughput (conn/sec)			
URL off length	AFPA (no cache)	AFPA on, (no cache) no rule	AFPA on, matching rule
11 char.	370.1	340.5	338.3
80 char.	361.5	321.9	319.4
160 char.	355.1	321.1	303.7

Table 6: Overhead of kernel mechanisms

Operation		Cost(μ sec)
TCP SYN policing	1 filter rule	7.9
	3 filter rules	9.6
classification and priority	1 rule	4.4
	3 rules	5.0
AFPA including URL parsing		19
URL-based rate control including URL matching	1 rule	5.0
	2 rules	5.8
	3 rules	6.5
URL-based priority including URL matching	1 rule	3.8
	2 rules	4.1
	3 rules	4.3

4.5.2 Discussion

The HTTP header-based rate control relies on sending TCP RST to terminate non-preferred connections as and when necessary. In a more user-friendly implementation we could directly return an HTTP error message (e.g., server busy) back to the client and close the connection.

Our current implementation of URL-based control handles only HTTP/1.0 connections. We are currently exploring different mechanisms for HTTP/1.1 with keep-alive connections to limit the number and types of requests that can be serviced on the same persistent connection. The experiments in the previous section have only presented results on URL based controls. Similar controls can be set based on the information in cookies that can capture transaction information and client identities.

4.6 Overhead of the Kernel Mechanisms

We quantify the overhead of matching URLs in the kernel for varying URL lengths. Table 5 shows that the overhead of matching a URL to a rule is moderate (under 6% for a 160 character URL). The throughput numbers are for 20 Webstone clients requesting an 8 KB file. Rules are matched using the standard string comparison (`strcmp`) with no optimizations; better matching techniques can reduce this overhead significantly. On a cache miss, the in-kernel AFPA cache introduces an overhead of about 10% for an 8 KB file. However, the AFPA cache under normal conditions increases performance significantly for cache hits. In our experiments we have the cache size set to 0 so that AFPA cannot serve any object from the cache. When caching is enabled Webstone received a throughput of over 800 connections per second on a cache hit.

Table 6 summarizes the additional overhead of the implemented kernel mechanisms. The overhead of compliance check and filter matching for TCP SYN policing with 1 filter rule is 7.9 μ secs. Simply matching the filter, allocating space to store QoS state, and setting the priority adds an overhead of around 4.4 μ secs for 1 filter rule. The policing controls are more expensive as they include accessing the clock for the current time. Surprisingly, the URL matching and rate control has a low overhead of 5.0 μ secs for a URL of 11 chars. This happens to be lower than SYN policing as the `strcmp` matching is cheaper for one short URL compared to matching multiple IP addresses and port numbers. The overhead of URL matching and setting priorities for a single rule is around 3.8 μ secs. The most expensive operation is the call to AFPA to parse the URL. AFPA not only parses the URL, but also does logging, checks if the requested object is in the network buffer cache, and pre-computes the HTTP response header.

5 Comparison of User Space and Kernel Mechanisms

In this section we compare the effectiveness of our kernel mechanisms with overload protection and service differentiation mechanisms implemented in user space. One might argue that kernel-based mechanisms are less flexible and more difficult to implement than mechanisms implemented in user space. User level controls although limited in their capa-

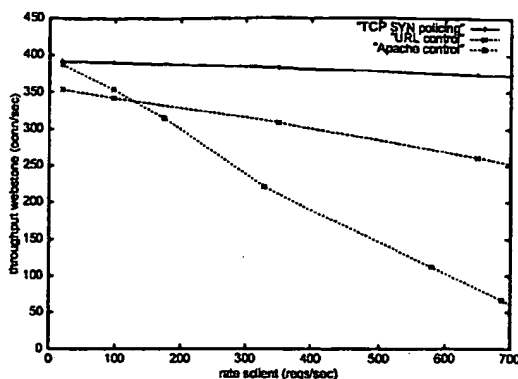


Figure 12: Throughput of kernel-based TCP SYN policing, kernel based URL rate control and Apache module based connection rate control. The throughput achieved by Webstone clients is measured against an increasing request load generated by sclient. The sclient requests are rate controlled to 10.0 req/sec with a burst of 2.

bilities, have easy access to application layer information. However, kernel mechanisms are more scalable and provide much better performance. In general, placing mechanisms in the kernel is beneficial if it leads to considerable performance gains and increases the robustness of the server without relying on the application layer to prevent overload.

To enable a fair comparison we have extended the Apache 1.3.12 server with additional modules [18] that police requests based on the client IP address and requested URL. The implemented rate control schemes use exactly the same algorithms as our kernel based mechanisms. If a request is not compliant we send a "server temporarily unavailable" (503 response code) back to the client and close the connection.

The experimental setup consists of a Webstone traffic generator with 100 clients requesting a file of size 8 KB along with an sclient program requesting a file of size 16 KB. The sclient's requests are rate controlled with a rate of 10 requests per second and a burst of 2; there are no controls set for the Webstone clients. During our experiments, we steadily increased the sclient's request rate.

Figure 12 illustrates that when the request load of the sclient program is low (20 reqs/sec), the Webstone throughput is 392 conn/sec and 387.3 conn/sec for TCP SYN policing and Apache user level controls respectively. These controls limit the sclient acceptance rate to 10.0 conn/sec. With in-kernel

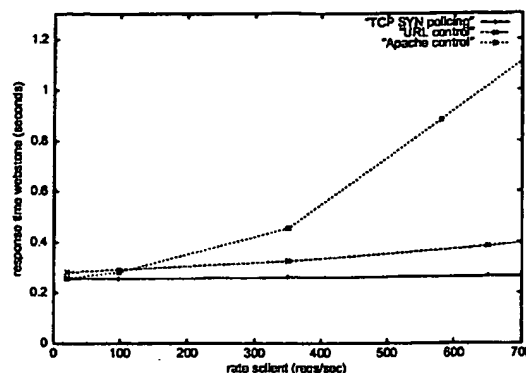


Figure 13: Response times using kernel-based TCP SYN policing, kernel based URL rate control and Apache module based connection rate control. The response time achieved by Webstone clients is measured against an increasing request load generated by sclient. The sclient requests are rate controlled to 10 req/sec with a burst of 2.

URL-based rate control the throughput is lower (354 conn/sec). This low throughput is caused by the additional 10% overhead added by AFPA (with no caching) as discussed in Section 4.6. As discussed earlier, with the cache size set to zero, we add more overhead than necessary for URL parsing, without the corresponding gains from AFPA caching.

As the sclient's request load increases further, TCP SYN policing is able to achieve a sustained throughput for the Webstone clients, while the Apache based controls shows a marked decline in throughput. The graph shows that for a sclient load of 650 reqs/sec the Webstone throughput for TCP SYN policing is 374 conn/sec; for in-kernel URL-based connection control it is 260.7 conn/sec; for Apache user level controls the throughput sinks to about 75 conn/sec. The corresponding results for response times are shown in Figure 13.

The experiment demonstrates that the kernel mechanisms are more efficient and scalable than the user space mechanisms. There are two main reasons for the higher efficiency and scalability: First, non-compliant connection requests are discarded earlier reducing the queuing time of the compliant requests, in particular less CPU is consumed and the context switch to user space is avoided. Second, when implementing rate control at user space, the synchronization mechanisms for sharing state among all the Apache server processes decrease performance.

6 Related Work

Several research efforts have focused on admission control and service differentiation in web servers [19], [20], [21], [22], [8] and [23]. Almeida *et al.* [8] use priority-based schemes to provide differentiated levels of service to clients depending on the web pages accessed. While in their approach the application, i.e., the web server, determines request priorities, our mechanisms reside in the kernel and can be applied without context-switching to user level. *WebQoS* [23] is a middleware layer that provides service differentiation and admission control. Since it is deployed in user space, it is less efficient compared to kernel-based mechanisms. While *WebQoS* also provides URL-based classification, the authors do not present any experiments or performance considerations. Cherkasova *et al.* [20] present an enhanced web server that provides session-based admission control to ensure that longer sessions are completed. Crovella *et al.* [24] show that client response time improves when web servers serving static files serve shorter connections before handling longer connections. Our mechanisms are general and can easily realize such a policy.

Reumann *et al.* [25] have presented virtual services, a new operating system abstraction that provides resource partitioning and management. Virtual services can enhance our scheme by, for example, dynamically controlling the number of processes a web server is allowed to fork. In [26] Reumann *et al.* have described an adaptive mechanism to setup rate controls for overload protection. The receiver livelock study [2] showed that network interrupt handling could cause server livelocks and should be taken into consideration when designing process scheduling mechanisms. Banga and Druschel's [27] *resource containers* enable the operating system to account for and control the consumption of resources. To shield preferred clients from malicious or greedy clients one can assign them to different containers. In the same paper they also describe a multi listen socket approach for priorities in which a filter splits a single listen queue into multiple queues from which connections are accepted separately and accounted to different principals. Our approach is similar, however, connections are accepted from the same single listen queue but inserted in the queue based on priority. Kanodia *et al.* [21] present a simulation study of queuing-based algorithms for admission control and service differentiation at the front-end. They focus

on guaranteeing latency bounds to classes by controlling the admission rate per class. Aron *et al.* [28] describe a scalable request distribution architecture for clusters and also present resource management techniques for clusters.

Scout [29], Rialto [30] and Nemesis[31] are operating systems that track per-application resource consumption and restrict the resources granted to each application. These operating systems can thus provide isolation between applications as well as service differentiation between clients. However, there is a significant amount of work involved to port applications to these specialized operating systems. Our focus, however, was not to build a new operating system or networking architecture but to introduce simple controls in the existing architecture of commercial operating systems that could be just as effective.

7 Conclusions and Future Work

In this paper, we have presented three in-kernel mechanisms that provide service differentiation and admission control for overloaded web servers. TCP SYN policing limits the number of incoming connection requests using a token bucket policer and prevents overload by enforcing a maximum acceptance rate of non-preferred clients. The prioritized listen queue provides low delay and high throughput to clients with high priority, but can starve low priority clients. We show that starvation can be avoided by combining priorities with TCP SYN policing. Finally, URL-based connection control provides in-kernel admission control and priority based on application-level information such as URLs and cookies. This mechanism is very powerful and can, for example, prevent overload caused by dynamic requests. We compared the kernel mechanisms to similar application layer controls added in the Apache server and demonstrated that the kernel mechanisms are much more efficient and scalable than the Apache user level controls.

The kernel mechanisms that we presented rely on the existence of accurate policies that control the operating range of the server. In a production system it is unrealistic to assume knowledge of the optimal operating region of the server. We are currently implementing a policy adaptation agent (Figure 4) that dynamically adapts the rate control policies to the changing workload conditions. This adaptation

agent uses available kernel statistics and past history to select appropriate values for the various policies and monitors the interaction between various control options on the overall performance during overload.

Our current implementation does not address security issues of fake IP addresses and client identities. We plan to integrate a variety of overload prevention policies with traditional firewall rules to provide an integrated solution.

References

- [1] "Cisco TCP intercept," <http://www.cisco.com>.
- [2] J. C. Mogul and K. K. Ramakrishnan, "Eliminating receive livelock in an interrupt-driven kernel," in *Proc. of USENIX Annual Technical Conference*, Jan. 1996.
- [3] P. Druschel and G. Banga, "Lazy receiver processing (LRP): a network subsystem architecture for server systems," in *Proc. of OSDI*, Oct. 1996, pp. 91-105.
- [4] O. Spatscheck and L. Peterson, "Defending against denial of service attacks in scout," in *Proc. of OSDI*, Feb. 1999.
- [5] "Ensim corporation: virtual servers," <http://www.ensim.com>.
- [6] "Alteon web systems," <http://www.alteonwebsystems.com>.
- [7] "Cisco arrowpoint web network services," <http://www.arrowpoint.com>.
- [8] J. Almeida, M. Dabu, A. Manikutty, and P. Cao, "Providing differentiated levels of service in web content hosting," in *Proc. of Internet Server Performance Workshop*, Mar. 1999.
- [9] T. Barzilai, D. Kandlur, A. Mehra, and D. Saha, "Design and implementation of an rsvp based quality of service architecture for an integrated services internet," *IEEE Journal on Selected Areas in Communications*, vol. 16, no. 3, pp. 397-413, Apr. 1998.
- [10] A. Mehra, R. Tewari, and D. Kandlur, "Design considerations for rate control of aggregated tcp connections," in *Proc. of NOSSDAV*, June 1999.
- [11] G.R. Wright and W.R. Stevens, *TCP/IP Illustrated, Volume 2*, Addison-Wesley Publishing Company, 1995.
- [12] Martin F. Arlitt and Carey I. Williamson, "Web server workload characterization: The search for invariants," in *Proc. of ACM Sigmetrics*, Apr. 1996.
- [13] P.Joubert, R. King, R.Neves, M.Russinovich, and J.Tracey, "High performance memory based web caches: Kernel and user space performance," in preparation.
- [14] "Khttpd - linux http accelerator," <http://www.fenrus.demon.nl/>.
- [15] "webstone," <http://www.mindcraft.com>.
- [16] G. Banga and P. Druschel, "Measuring the capacity of a web server," in *Proc. of USITS*, Dec. 1997.
- [17] "apache," <http://www.apache.org>.
- [18] L. Stein and D. MacEachern, *Writing Apache modules with Perl and C*, O'Reilly, 1999.
- [19] T. Abdelzaher and N. Bhatti, "Web server qos management by adaptive content delivery," in *Int. Workshop on Quality of Service*, June 1999.
- [20] L. Cherkasova and P. Phaal, "Session based admission control: a mechanism for improving the performance of an overloaded web server," Tech. Rep., Hewlett Packard, 1999.
- [21] V. Kanodia and E. Knightly, "Multi-class latency-bounded web servers," in *Intl. Workshop on Quality of Service*, June 2000.
- [22] K. Li and S. Jamin, "A measurement-based admission controlled web server," in *Proc. of INFO-COMM*, Mar. 2000.
- [23] Nina Bhatti and Rich Friedrich, "Web server support for tiered services," *IEEE Network*, Sept. 1999.
- [24] M. E. Crovella, R. Frangioso, and M. Harchol-Balter, "Connection scheduling in web servers," in *Proc. of USITS*, Oct. 1999.
- [25] J. Reumann, A. Mehra, K. Shin, and D. Kandlur, "Virtual services: A new abstraction for server consolidation," in *Proc. of USENIX Annual Technical Conference*, June 2000.
- [26] H. Jamjoom and J. Reumann, "Qguard:protecting internet servers from overload," Tech. Rep., University of Michigan, CSE-TR-427-00, 2000.
- [27] G. Banga, P. Druschel, and J. Mogul, "Resource containers: a new facility for resource management in server systems," in *Proc. of OSDI*, Feb. 1999.
- [28] M. Aron, D. Sanders, P. Druschel, and W. Zwaenepoel, "Scalable content-aware request distribution in cluster-based network servers," in *Proc. of USENIX Annual Technical Conference*, June 2000.
- [29] D. Mosberger and L. L. Peterson, "Making paths explicit in the scout operating system," in *Proc. of OSDI*, Oct. 1996, pp. 153-167.
- [30] M. B. Jones, J. S. Barrera III, A. Forin, P. J. Leach, D. Rosu, and M. Rosu, "An overview of the Rialto real-time architecture," in *ACM SIGOPS European Workshop*, Sept. 1996, pp. 249-256.
- [31] Thiemo Voigt and Bengt Ahlgren, "Scheduling TCP in the Nemesis operating system," in *IFIP WG 6.1/WG 6.4 International Workshop on Protocols for High-Speed Networks*, Aug. 1999.

TRIAD: A New Next-Generation Internet Architecture

Computer Science Department
Stanford University
{cheriton,mgritter}@dsg.stanford.edu

Abstract

Today, the primary use of the Internet is content distribution — delivery of web pages, audio and video streams to client browsers. However, scaling to meet the enormous demands of the web have required *ad hoc* and, in some cases, proprietary protocols and mechanisms to be deployed. Unfortunately, these *ad hoc* mechanisms have scaling problems and conflict with the original Internet architecture. IPv6, the current leading candidate for a next generation Internet architecture, provides more addresses but does not help with the content problem, given that its design predates the web.

In this paper, we present TRIAD as a new next generation architecture. A key aspect of TRIAD is the explicit inclusion of a *content layer* that provides scalable content routing, caching and content transformation. TRIAD also provides extensible path-based addressing using a simple “shim” protocol on top of IPv4. We claim that TRIAD not only provides scalable content distribution, but also solves the Internet problems with supporting *network address translation (NAT)* and provides innovative solutions to mobility, virtual private networks, policy-based routing and source spoofing. Its compatibility with IPv4, TCP, DNS and other dominant Internet protocols facilitates incremental deployable.

1 Introduction

With the emergence of the web, the primary use of the Internet is content distribution, i.e. web pages, and increasingly audio and video streams. Some measurements indicate that 70 to 80 percent of Internet traffic is HTTP traffic (and most of the rest of the traffic is routing and DNS). That is, almost all of the traffic in the wide area is delivery of content, and ancillary traffic to locate it and determine how to deliver it. Today, millions of clients are accessing thousands of web sites on a daily basis, with the top 20 web sites supplying about 10 percent of the content. Moreover, new popular web sites and temporarily attractive web sites can prompt the arrival of a so-called *flash crowd* of clients, often overwhelming the resources of the associated web site.

To scale content delivery to support these demands, a variety of *ad hoc* and, in some cases, proprietary mechanisms have been deployed. For instance, content is geographically replicated at multiple sites with specialized name servers that redirect DNS lookups to nearby (to the client) sites based on specialized routing, load monitoring and Internet “mapping” mechanisms, so-called *content routing*. Further, proxies in the network provide transparent caching of content, further reducing the load on web sites and the network resources between the cache and the web site. Finally, load-balancing switches at each web site allow the *virtual host* for the web site to be realized as N physical hosts, distributing the content requests across these hosts based on individual server load and the content it holds. Future extensions are expected to provide content transformations proxies that will transform content to a representation suitable for the requesting client, such as a mobile PDA.

These mechanisms violate the original Internet architecture in various ways, do not fully address scalable content distribution, and compromise the basic philosophy of the Internet as being based on open, community-based standards. In particular, content routing requires a DNS server that accesses some form of routing information, a layer violation (or duplication of the routing layer), yet still requires the world-wide clients of a site to access a single centralized server as part of accessing that site. This roundtrip to a central server is quickly becoming the dominant performance issue for clients as Internet data rates move to multiple gigabits, reducing the transfer time for content to insignificance. Transparent caching requires hijacking

transport-level connections, violating the end-to-end semantics and causing connection failures when the routing changes to route around the cache. Moreover, these caches introduce extra delay in connection setup for non-cachable content while they collect the HTTP header information required to determine whether the content is cachable or not. Load balancing switches rely on *network address translation (NAT)* which requires rewriting addresses, port numbers, transport-layer checksums and even packet data, making mockery of the original end-to-end semantics. (Network Address Translation [11, 12] (NAT) is also being widely deployed for other reasons, such as *address allocation autonomy*, multi-homing, concealing endpoints and increasing the number of addresses available in an edge network.

In 1992, work on IPng, next-generation IP, was initiated based on concern that the Internet was running out of addresses. This effort, predating the web, focused on providing more addresses but failed to anticipate the strong emergence of content as the primary use of the web and thus failed to address the content distribution issues described above.

In this paper, we present TRIAD¹, as a new next generation architecture. TRIAD defines an explicit *content layer* that provides scalable content routing, caching, content transformation and load balancing, integrating naming, routing and transport connection setup. In particular, all end-to-end identification in TRIAD is based on names and URLs, with IP addresses reduced to the role of transient routing tags. At this content layer, The integrated directory, routing and connection setup to provide efficient content routing to replicated content and eliminate roundtrip times to access the content in most cases. Finally, TRIAD supports path-based addressing using a simple "shim" protocol on top of IPv4 called WRAP, for content routing control and extensible addressing. This extensible addressing eliminates the need to transition the Internet to IPv6. We claim that TRIAD not only provides scalable content distribution, but also solves the Internet problems associated with *network address translation (NAT)*. TRIAD also provides attractive solutions to mobility, virtual private networking, policy-based routing and source spoofing. TRIAD can be incrementally deployed, initially without changes to end-hosts or applications beyond that already required for NAT.

The next section describes the TRIAD content layer. Section 3 describes the named-based identification of endpoints and its implications in more detail, illustrating with the associated modifications to TCP. Section 4 describes the integrated naming and routing in TRIAD. Section 5 describes the extensible path-based addressing using WRAP, its expected implementation and WRAPsec, an IPsec-like mechanism for end-to-end security. Section 7 discusses the implementation of TRIAD and some measurements to evaluate it we have made to date. Section 6 describes the TRIAD approaches to mobility, VPNs, policy-based routing and extended forwarding checks. Section 8 describes an extended NAT router that allows incremental deployment of TRIAD. Section 9 describes the incremental deployment of TRIAD. Section 10 describes related work, and we close with conclusions.

2 TRIAD Content Layer

At the content layer, a client either requests to access some content, identified by some name specification, to read or to write (or both). For example, a client may request the CNN news, which is delivered as some combination of web pages and audio and video streams. In contrast, lower levels of the architecture simply transmit or receive data packets.

The TRIAD content layer may return a *network pointer* through which content may be read or written, rather than the content itself, especially when the content is large or indeterminate in length.

For compatibility with the World-wide Web, the TRIAD content layer uses the Web Uniform Resource Locator (URL) as the format for content identification, optionally augmented with web "cookies". Further, this "network pointer" is realized as an HTTP/TCP connection through which the content is read or written.

The content layer is implemented by *content routers* that direct the requests towards content servers storing the content, *content servers* that provide the content services, and *content caches* that transparently store some content nearer to the client than the servers themselves. It may also include *content transformers* that transform the content from one form to another in response to characteristics of the client and its network connection. For instance, a content transformer may reformat content for presentation on a PDA, and transmission to this PDA over a limited bandwidth wireless link.

¹ TRIAD is an acronym, standing for *Translating Relaying Internet Architecture integrating Active Directories*.

For example, in TRIAD, a client sends a lookup request with a URL for the CNN news page to its content router. The content router looks up this URL (typically just the DNS portion of the URL) and determines a nearby replica for this content, forwarding the request to a next hop accordingly. The next hop content router looks up this URL as well and may determine it has this content cached locally. In this case, it returns TCP connection information to the client, allowing the client to read this content from its cache over this (HTTP/TCP) connection.

In this way, the content layer explicitly supports the key requirements for scalable content distribution. In particular, it explicitly supports replicated content, leveraging the routing information to locate the nearest replica. It further provides hierarchical transparent caching, minimizing the implosion of demand on popular content servers and their Internet connections without requiring explicit configuration of proxies in the browsers while still avoiding the "route-around" danger with current transparent caches. Finally, it eliminates a roundtrip relative to the current Internet by combining the name lookup with connection setup.

The TRIAD content layer also supports multicast-based content distribution, by allowing a content lookup to subsume the functionality of the current multicast subscription, connecting client into the multicast session and the associated distributed tree.

For simplicity, one can view the DNS portion of a URL as mapping to the content server and the file name portion selecting the file within the content server. However, there is some flexibility in this partitioning of a URL into DNS name and file name portion. For instance, a prefix of the DNS name can determine a content volume within a specific content server. Similarly, a content router can direct an HTTP connection to a separate server based on the file portion. The primary constraint now is that the file portion is not available until the HTTP connection is established. For simplicity, we describe TRIAD assuming the conventional use of DNS name specifying the content server/volume, and file name portion specifying the content within this volume.

For deployability, TRIAD is designed to be highly compatible with existing Internet infrastructure, requiring extensions primarily at the directory system level. The TRIAD content layer subsumes the DNS directory system, providing DNS name lookup as a subset of its services. It also includes the wide-area routing system, allowing it to map content name to the closest replica based on the proximity information (accessed from the routing system). Finally, the content layer includes a mechanism for creating the so-called network pointers, more conventionally known as transport connections, thus subsuming the connection setup portion of TCP. The TRIAD content layer interfaces to the conventional transport and (inter)network layers of the Internet, allowing IP and TCP to be used, but with some modifications.

The following sections describe specific changes in more detail.

3 Named-based Identification Endpoints

In TRIAD, endpoints are identified by name. At the content level, the endpoint is a source of content and is identified by a URL. At the host interface level² the endpoint is identified by a hierarchical character-string (DNS) name. For example, "pescadero.stanford.edu" identifies a host interface on a host at Stanford. In contrast, an address for this interface may change over time, even during the lifetime of transport layer connections.

This *name* is the basis for all end-to-end identification, authentication, and reference passing. There is no other identifier that is global and persistent, unlike addresses in IPv6 and in the original Internet architecture. (IPv4) addresses serve as *routing tags* and have no end-to-end significance.

A multicast channel as in the EXPRESS single source multicast (SSM) model [4] is identified by a name subordinate to the name of the source host. For example, "chan1.pescadero.stanford.edu" could be the name of a multicast channel with "pescadero.stanford.edu" as the source. (TRIAD only supports the single-source model of multicast.)

3.1 Name-based Transport Connection Setup

In TRIAD, a transport connection is setup by a name lookup and connection setup protocol at the content layer called *Directory Relay Protocol (DRP)* [5]. The client sends a lookup request containing a URL and possibly other information, such as "cookies" to the content layer's directory system. For brevity, we refer to this information generically as the "name", rather than detailing URL and cookies each time. The directory

²Packets are sent to a host interface, not to a host, the same as the original Internet architecture.

system routes the name request through to a content server able to provide this named content, if any, which then either returns the content directly or else sets up transport connection state and returns the associated connection information to the requesting client.

DRP serves as a connection setup protocol for TCP, carrying the same sequence number, port and option information as a TCP SYN packet along with the content identification. The content identification is stored with the connection state and is used to re-bind the connection to new addressing if the connection is failing to get packets through. In the expected case, this rebinding maps to the endpoint that is storing the transport-level state of the connection, allowing the transport-level connection to continue with the new address binding, similar to an ARP-level rebinding. In particular, the rebinding uses the canonical name for the content server so it first attempts to rebind to the same replica. When the original content server is not reachable, a end-to-end name lookup can bind to a new server, restarting the content access from the beginning.

Using DRP, a TRIAD TCP connection to content can be established with a single round-trip from client to server. This contrasts with the two roundtrips commonly required with the current DNS/TCP separation. Moreover, a transparent content cache can intercept a DRP request and service this content request without having to hijack the transport packets because, at the packet-level, the packets are addressed to the transparent cache. This avoids the connection failures that can occur currently when packets are rerouted around a transparent cache. The transparent cache can also receive all the HTTP information in the DRP request, eliminating the need to incur an extra roundtrip to get this information, as with the current TCP/HTTP setup behavior. Also, TRIAD TCP can transparently recover from changes to the addresses used to reach the endpoints, whether caused by intermediate node timeouts or reboots or link failures (assuming an alternate path is available). This rebinding makes the translation in the network effectively *soft state*, preserving one of the key properties of the Internet. UDP-based services are expected to similarly rebind from the name, either periodically or on timeout, in the case of a reliable protocol built on UDP. Finally, DRP carries the client name and identification, allowing the content server to determine this information without callback or reverse name lookup. (The network can validate the DNS-level identification as part of forwarding the request.)

This same name-based connection setup with DRP works for joining a multicast session as well. The name lookup is sent to the source, which returns the required information to join the multicast session, setting up requisite multicast state at the network layer along the path.

For backwards compatibility with current Internet, TRIAD hosts also support (and fail-over to) conventional TCP connection setup when communicating with unmodified IPv4 hosts.

3.2 Name-based Transport Pseudo-Header

In TRIAD, the transport layer checksum covers the name of the source and the name of the destination and *does not* include the packet address. In the case of multicast, the pseudo-header is based on the name of the channel. The addressing allows efficient forwarding of the packet to a destination; the name-based pseudo-header ensures it arrived at the correct destination (even though the names are not present in the header). Thus, a transport connection is between two endpoints identified by name, not address.

A transport-level checksum based on this pseudo-header provides end-to-end reliability because it detects corruption of the packet addresses in transit yet does not need to be modified in transit as part of relaying (or forwarding) the packet even though the packet addresses are modified.

This name-based pseudo-header checksum also allows end-to-end security with NAT because changing the addresses does not require updating the checksum. TRIAD includes a security layer similar to IPsec, WRAPsec, which uses names for identification and the same pseudo-header for integrity check verification (ICV), and provides end-to-end security. Note that dispatch to connection state before validating the checksum is required in TRIAD for both secure and insecure connections, unifying the processing in both cases. With conventional TCP implementations, the checksum is often checked before mapping to the associated TCP connection state.

On connection setup, the local endpoint computes and saves a *precomputed pseudo-header checksum value (PHCV)* based on the name of the source and the destination, similarly at the remote endpoint. The source and destination names are also stored in the connection state record with the PHCV. When a packet is transmitted, the checksum is computed starting with the PHCV, effectively incorporating this name-based pseudo-header into the packet checksum. On reception, the packet is demultiplexed to the TCP connection

state based on the packet addresses and port numbers. This receiving connection state contains the same PHCV, allowing the receiver to (re)compute the packet checksum efficiently. If the computed checksum fails to match that in the packet, the packet is considered corrupted in transmission.

If the packet does not map to a valid connection state, the receiver does a reverse name lookup to determine the source name and looks up the connection by name. If the name lookup fails, the packet is discarded as a corrupted packet. An endpoint may receive a packet for an existing connection that does not match based on addressing (perhaps because of a reboot of an intermediate node). The name-based mapping allows the connection state to be located and the address mapping to be corrected.

To allow connection setups with minimal modifications to current TCP clients, TRIAD-TCP includes a (new) option that carries the PHCV in the SYN packet, rather than relying on DRP. For backward compatibility, TRIAD-TCP can also use the current TCP checksum algorithm for a connection where the source and destination are in the same realm.

TRIAD-TCP provides a negotiated "unreliable" mode, which simply disables retransmissions. This minor extension to TCP as a negotiated option allows applications such as real-time VoIP and video to use TRIAD-TCP and automatically get the rebinding behavior described above (as well as the TCP congestion avoidance mechanisms). With this provision, TCP can replace the wide-area use of UDP in all cases that we are aware of. Then, UDP is only used local to a realm, if at all.

The behavior in TCP of allowing infinite timeouts when the connection is idle is supported in TRIAD by a timestamp on the name stored in the connection record, and rebinding when a connection becomes non-idle if the connection has been idle with no relookup for some excessive period of time.

TRIAD routers include support for WRAMP³, an ICMP-like protocol for sending "destination unreachable" messages, similar to ICMP, thus informing hosts (on a best efforts basis) when the addressing is no longer valid.

With these changes, applications on WRAP-aware hosts using TCP have end-to-end semantics and end-to-end reliability, and are oblivious to loss of intermediate translating state except possibly for the performance impact. These changes to TCP do not change the packet format, are local to the implementation, and allow unmodified hosts to communicate within a realm.

3.3 Aliases and Content Routing

Content routing is supported in TRIAD by associating an *alias name* with the content. The DRP lookup of an alias name is mapped to the associated canonical name that is closest to the requesting client. (An alias can represent a host or a set of hosts.) For example, "yahoo.com" can be an alias for a set of canonical names "yahoo1.com", "yahoo2.com", etc. representing the set of content servers for Yahoo. For example, a lookup of "yahoo.com" might select "yahoo3.com", causing the DRP request to be routed towards this specific server. If this content server does not respond, the first-hop content router can select a different content server. The client can also request exclusion of a non-responsive server in a DRP request.

In this fashion, client requests are routed over the best path to the desired content in the normal case yet can recover from a failing content server. In this sense, TRIAD provides an "anycast" capability at the directory level with network and client control to reselect alternatives based on its direct experience with the chosen server. In contrast, conventional anycast mechanisms at the network layer, such as that provided by IPv6, may repeatedly route anycast packets to a server that is not functioning adequately, based on a higher-level (than network layer) evaluation.

This named-based approach relies on the name mapping being as reliable and as secure as packet forwarding, so applications can use names without loss of reliability or security. It also requires that the directory service have sufficient network routing information to determine the proximity of the interfaces identified by the alias, relative to the requesting client. This is supported in TRIAD by integration of the naming and routing. (Of course, higher-level checksumming, encryption, and authentication can provide additional security and reliability when needed.)

4 Integrated Naming and Routing

In TRIAD content layer, naming and routing are integrated in three senses. First, the router implements name lookup service, rather than a separate server that is off the data path. Second, a name lookup can

³The Wide-area Relay Addressing Management Protocol

return a route or *path specification* to the client. Finally, the routing mechanisms and directory mechanisms are strongly coupled in the router: the routing table maps from name to next-hop, rather than mapping address to next-hop, the routing information identifies endpoints and next-hop nodes by name, and name mapping information is disseminated by routing advertisements throughout the network. Individual hosts can participate in the routing by indicating the names or *content volumes* they support and the "distance" to that content, indicating the cost of reaching that content through them.

In practice, it seems unnecessary and excessive management overhead to actually have every router involved in the directory service⁴. In TRIAD, the key routers involved in the directory service are the firewall/NAT routers between realms and BGP-level routers between autonomous systems. We collectively refer to routers involved in the integrated naming and routing system by the term *content router (CR)*. We use the term *content resolving router (CRR)* for a router that just participates as a DNS resolver. This, an ISP would typically have a small number of CRs, corresponding to its current BGP routers, plus conventional routers within one realm, obvious to the content layer.

Each CR or CRR acts as a name server for each realm to which it is directly connected. For names in the same realm as the requester, the TRIAD directory service behaves exactly the same as current DNS for IPv4 clients making DNS lookup requests. That is, a DNS request with *QTYPE = A* simply returns the IPv4 address of the associated local host, as determined by the local name database. In particular, a content router can use name lookup to locate other CRs in the same ISP (and thus presumably connected to the same ISP realm). For inter-realm lookups, the CR may return a similar local IPv4 address that it translates to the remote destination address or it may return an *address path*, as supported by the WRAP protocol. For now, consider an address path as a sequence of addresses, typically spanning several address realms, designating a loose source route to the packet destination. (Like *split DNS* in NAT, different realms have different paths associated with the same name.) This protocol is described in more detail in Section 5.

Typically, a firewall router participates as a CRR in the ISP realm to which it connects. A name lookup request it receives from a client in its private realm for a name outside of this realm is passed to a content router in the ISP, which returns an address path for a content server to the firewall. The firewall then modifies this address path information by adding its addressing information as appropriate and passes the result back to the client. Thus, each such client of the ISP behaves similar in name lookups and routing participation to that of a conventional NAT router.

4.1 DRP Implementation

A name lookup is performed by recursively relaying the DRP request across CRs from the requestor to an content server for the specified content. At each node, the name request is logically relayed along the path that packets are to take, based on local knowledge of which peer is the "next hop" towards the requested content. After the request reaches an appropriate content server, a response is returned along the reverse path through the CRs, with each one modifying the addressing to finally produce the address path suitable for the requestor to use.

By relaying the name lookup request across the same path as the packets are to flow, any necessary forwarding state can be set up in intermediate CRs. Also, this means that DNS is as available as endpoint connectivity, i.e. if the endpoint is reachable, name lookup to that endpoint works. Moreover, the trust in name lookup matches the trust in delivery because both depend on the same set of network nodes. Also, the name lookup load for a path is imposed just on the routers on that path so upgrading a router on that path in data capacity can also upgrade the name lookup capacity on that path. Moreover, transparent caches, can added to a loaded path, and off-load name lookup and content delivery from subsequent portions of the path. Thus, one can balance data throughput and directory service capacity, similar to how this is handled with file systems. Consequently, in TRIAD, you can rely on names as much as you rely on addresses in the current Internet architecture, and name lookup capacity scales with the Internet.

A CR can also provide reverse (i.e., address-to-name) lookup by forwarding a reverse lookup request along the same path as a packet with the same address.

In a multi-homed realm, such as an enterprise network served by two ISPs, the internal naming and routing selects one of the CRRs for the name lookup based on local routing information. This mechanism is

⁴For instance, within one realm, any of the routers supporting a directory service is reachable by address without the directory service, allowing a client to access this service if there is connectivity to it.

also expected to detect when the selected node fails or becomes disconnected, causing traffic to be rerouted through the other CRR. Because the name is the primary identifier and can be rebound without losing the connection state, the connection can survive this redirection to the other router similar to a connection surviving rerouting in the current Internet. With routing updates significantly damped in the Internet to avoid oscillations, especially at the BGP level, we expect TRIAD name rebinding to provide recovery latency that is comparable, if not superior, to that of the current Internet.

4.2 Name-based Routing

The TRIAD *Name-Based Routing Protocol (NBRP)* [3] performs routing by name with a structure similar to BGP. Just as BGP distributes address prefix reachability information among autonomous systems, NBRP distributes *name suffix* reachability among address realms. Routing information is distributed among CRs and maintained locally with next hops and destinations specified in terms of names and name suffixes. With this step, the name directory and the routing table logically become a single entity, reducing the overall complexity of the CR software⁵.

This *name-based routing* distributes name mapping information to ensure its availability, to distribute the name lookup load, and provide faster name lookup response. Identifying endpoints by name is also necessary because addresses are not unique across the multiple realms in TRIAD. (Intra-realm routing can use existing routing protocols. Intra-realm reliability of name service can be ensured by duplicate servers as now.)

The key challenge with name-based routing is maintaining the routing database efficiency in the presence of names that do not match the routing hierarchy. To reduce routing table size to a feasible level, name-level redirection mechanism is used to handle hosts whose names do not match network topology. For example, all hosts with Harvard names not in the same address realm as the authoritative server for Harvard.EDU would have redirect records at that server. Consequently, the routing database only needs to deal with Harvard.EDU, not hosts subordinate to this domain. Nevertheless, TRIAD can deal with third-level names as necessary, such as europe.ibm.com, japan.ibm.com, etc.

NBRP also supports combining collections of name suffixes that map to the same routing information into *routing aggregates*. For instance, we expect an ISP relay node to group all of the names from its customers into a small number of aggregates. With aggregates, routing updates typically update a small number of aggregates rather than the large number of individual name entries contained in each aggregate. Moreover, all names in a routing aggregate may be treated identically in routing calculations, thus reducing load at CRs. Aggregate membership should be relatively long-lived, so that CRs can amortize the cost of learning the aggregate membership over many routing updates. Section 7 describes measurements of a preliminary evaluation of the benefit of route aggregates.

Although TRIAD name lookups may contain a full URL, cookies and other content level identification, the key information maintained by network nodes is essentially the same as current DNS. Most more detailed content information is maintained at end nodes, and cached in transparent proxy caches.

To keep the number of NBRP neighbors small so that the routing overhead is acceptable, nodes on the interior of a realm can be added that perform only route updates and name lookups but do not otherwise participate in the routing. Current BGP speakers could be upgraded to perform as NBRP speakers as well participating in the routing. The CRR of a typical ISP customer is a degenerate case of this approach.

4.3 Host Advertising, Aliases and Content Routing

A host can advertise an alias associated with its canonical name(s) to the NBRP system together with the hop count cost of accessing the associated content through this host. This cost is in terms of "hop-equivalent" response time units. That is, if the response time cost of going an extra hop is estimated as 2 millisecond, this advertised cost is k if the server is averaging $2 * k$ milliseconds to respond to a request. The routing system adds this cost to the routing cost of accessing this host as it disseminates the naming and routing information. Explicit advertising of host cost is limited to content servers, a very small fraction of the hosts on the Internet.

A CR receiving advertisements of two or more canonical names with the same alias name groups these canonical names for lookup under this alias. On receipt of a name lookup for the alias, the CR orders

⁵Note that a conventional routing table is a simple directory: It is queried with an IP address to determine the forwarding information. With TRIAD, the equivalent directory in a relay node is queried using the DNS name.

the associated canonical names by proximity (determined from the routing database) to the requestor and forwards the request to the closest proxy.

This approach has several benefits for content routing. Because this information is pushed out to each CR, a client request is immediately routed to a close-by content server that than having to first go to a distant central server. This distribution also avoids excessive load and failure-dependence on a central resource. Moreover, the proximity information is based on server load and availability, not just network access.

4.4 Security

The directory service supports message authentication using public-key and shared-key cryptographic signatures. This allows clients to determine that the answer they get from the directory service is authentic, and allows relay nodes to identify a particular principal associated with a client.

NBRP updates are authenticated by cryptographically signing "delegations" of part of the namespace to a CR's peers, in a manner similar to Secure BGP [18].

Unlike DNS security[7], a single name-to-address mapping cannot be signed by the authoritative server for a name because the address also depends on the intervening CRs. Instead, CRs must establish trust relationships.

5 WRAP and Path-based Addressing

In TRIAD, WRAP, the *Wide-area Relay Addressing Protocol (WRAP)*, is a "shim" protocol that specifies, together with the IP packet source and destination addresses, a *path* to desired content. It carries the transport header and data as its payload, similar to other IP encapsulation protocols.

The WRAP header contains a pair of *Internet Relay Tokens (IRTs)*, the *reverse token* and *forward token*. The forward token represents the path the packet is to take and the reverse token indicates the path the packet has taken to this point. An IRT is a potentially opaque variable-length field that specifies a path from the source to the destination. It may also be simply a sequence of IPv4 addresses.

A WRAP packet is formatted as in Fig. 1 as the payload of an IPv4 packet.

0-7	8-15	15-23	24-31
protocol	length	foffset	reserved
reverseToken			
forwardToken			
data			

Figure 1: WRAP Packet Format

The protocol field specifies the higher-layer protocol in the "data" field using the same types as for IP, e.g 6 for TCP. The length field is the number of 32-bit *components* in the header. Thus, the WRAP header length in octets is $4 + \text{length} * 4$. The components specify the reverse and forward tokens. The foffset field is an offset into the list of components where the forwardToken starts, with 0 referring to a null reverseToken, so the forward token starts in the first 32-bit field. The forwardToken is used by the node addressed in the IP packet destination address to determine how to relay the packet to its destination. The reverseToken is a value used by the node addressed by the IP packet source address to refer to where the packet came from.

The data field contains a TCP, UDP, or other transport protocol packet.

A WRAP source sends packets to a destination by forming an IPv4 packet with the IP destination address set to the address of the next relay, the WRAP header containing the IRT in the forwardToken of the destination relative to this relay node, and a null reverseToken. Thus, the length field is the length of the forwardToken and the foffset field is 0. (The foffset value is also the length of the reverseToken.)

A node, on receiving an WRAP packet:

1. maps the (SA,DA) of the packet to a *virtual interface (VI)* that represents the local endpoint of the realm "channel" on which the packet arrived. It maps the next k 32-bit components of forwardToken

field (assuming offset is less than length) to a corresponding relay entry in a relay table associated with this VI.

2. determines from this entry the next IP source address (the "egress" interface), the next relay's IP address, the new forward token, and the rewrite of the reverse token to perform.
3. forwards the modified packet to the next relay node, with the IPv4 destination address as that of this next relay and the IP source address determined above, and increments the offset field.

Each relay node thus "consumes" one or more 32-bit components from the forwardToken and adds an equal number of components to the reverseToken. (However, both these fields may be translated according to the information in the relay node's lookup table.) The reverse token, when component-reversed, must be recognized by this node when used as a forward token, to send packets back toward the source.

Normally, a node just rewrites the first forward token component and increments the offset by 1. In the simplest case, the rewritten component is just encodes the IP source address of the incoming packet (that is, the last relay point.) This restricted form of relaying, though less general than WRAP allows, is more amenable to hardware implementation, because less of the packet needs to be rewritten.

If the node does not recognize the forward token, it drops the packet and may send a WRAMP message back to the previous node. The relaying state may include filters on sources from which to accept packets and destinations allowed for given sources.

The receiver of a WRAP packet is a node that receives the packet with a null forwardToken, or receives a packet with a multicast destination and subscribes to that multicast source.

The actual source of the packet is identified by the reverseToken and the IP source address. The receiver can contact this previous relay identified by the IP source address to do a reverse name lookup on this IRT to determine the name of the actual source.

With WRAP, a packet is reassembled from fragments at each intermediate relay node, because each is a destination from the IP standpoint. This feature reduces the risk of carrying packet fragments all the way to the destination only to discover some fragment is missing. Each WRAP node sets the TTL of a WRAP packet according to its estimate of hops to the next relay. Packets cannot loop at the WRAP level because some non-zero portion of the WRAP IRT is consumed at each relay node.

Backbone or wide-area ISPs can connect at peering points, the same as today, but with high-speed relay routers at these points. At this level, the firewall or border router is extended to act as a TRIAD relay between realms, translating packet addresses as it relays packets between the realms that it interconnects.

Between the IP addressed nodes, a packet is routed by the normal IPv4 routing protocols used within the realm. Thus, WRAP is similar to loose source routing with the relay nodes as the designated nodes on the path it is to follow.

Within a realm, the operation of naming, addressing and routing operates the same as currently with IPv4. Thus, there are no host or router changes required. A packet that does not travel outside of a single address realm can omit the WRAP header entirely.

5.1 WRAP Example with Multiple NAT Realms

Fig. 2 illustrates the operation of TRIAD between realms with two hosts, `src.Harvard.EDU` and `dst.Ietf.ORG`, assuming `Harvard.EDU` and `Ietf.ORG` are two separate realms connected via a single intermediate realm, the "external" Internet. (For simplicity, we illustrate just with DNS names, not full URLs.) For `src` to send to `dst`, the name lookup of `dst.Ietf.ORG` is handled by the relay node `relay.Harvard.EDU` for this realm, with internal IPv4 address `RA1` and external IPv4 address `RA1'`. This relay determines the appropriate next relay from its directory mapping of `Ietf.ORG` and then communicates the name lookup across the Internet to the relay `relay.Ietf.ORG`, the relay for the `Ietf.ORG` realm. (This relay has internal IPv4 address `RA2` and external IPv4 address `RA2'`.) In response to this query, `relay.Ietf.ORG` communicates with `dst` to set up connection state and then returns to `relay.Harvard.EDU` an IRT `f'` that designates `dst` relative to `RA2'` and the associated transport connection information. Then, `relay.Harvard.EDU` returns an IRT `f` to `src` which designates `dst.Ietf.Org` relative to `RA1`, creating any state it needs to map `f` to `f'`, passing the transport connection information through.

Then, `src` sends the first data packet over this connection as an IPv4 packet addressed to `RA1` with `f` stored in the WRAP header. On reception, `RA1` translates `f` into `f'` and transmits the packet with

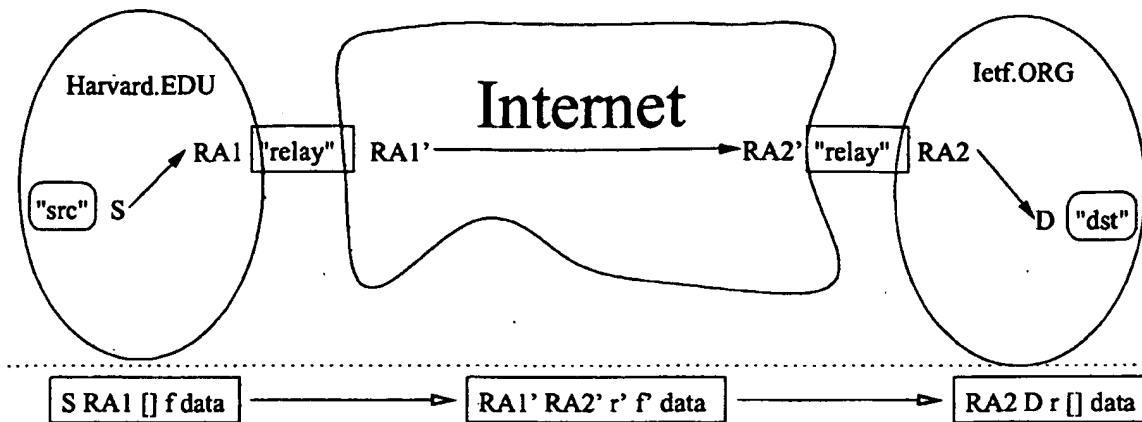


Figure 2: Inter-realm packet transmission in TRIAD: The host named "src" with IPv4 address S in realm Harvard.EDU sends to the host named "dst" and IPv4 address D in realm Ietf.ORG. The packets below the dotted line indicate how the IP and WRAP headers changes as it crosses the three realms, with the header listed as source address, destination address, reverse IRT, and forward IRT.

destination address $RA2'$ and source address $RA1'$, as shown in the middle packet in the figure. The WRAP header also contains the reverse IRT r' , which indicates the source of the packet relative to $RA1'$.

On reception at $RA2'$, the reverse IRT in the packet is translated to a new value r which represents src.Harvard.EDU relative to $RA2$. This relay then transmits the packet with an empty forward IRT, IPv4 destination address D and source address $RA2$, as shown in the rightmost packet in the figure.

Thus, dst receives the packet as a normal IPv4 packet sent by its relay $RA2$ but also containing an IRT that identifies the actual source of the packet relative to its relay. The packet is then passed up to the next higher layer processing, such as TCP or UDP.

The destination can respond to a packet directly by "reversing" the IRT and sending the packet to the local relay with this reversed IRT. This causes the packet to return along the reverse of the relay path on which the original packet was received. Alternatively, the destination can perform a lookup on the source name to get a separate IRT and RA to reach this host. This alternative is more flexible, allowing for asymmetric routing at the cost of an extra name lookup.

An address realm at the leaf level may correspond to an enterprise or university network, a military installation, or much smaller units like a collection of autonomous sensors or a home network or even a set of virtual hosts on a single physical machine. Higher-level address realms correspond to local and global Internet service providers (ISPs).

The IRT and the relay address are local in scope and transient. That is, the IRT is only meaningful relative to the relay and realm and is only guaranteed to be *T-stable*: it does not go from one valid association with a relay to another in less than time T , where T is typically hours. In particular, it can become invalid at any time but can only be reassigned to another use after time T . Thus, passing an IP address or an IRT in the data portion of a packet to the other endpoint is meaningless in general.

A WRAP proxy, referred to as a WRAPID gateway (see Section 8), allows existing IPv4 hosts to interact with WRAP-enabled hosts and servers without any modification. A WRAPID gateway is just an extended NAT-capable router or firewall which is able to WRAP and unWRAP packets going through it, as appropriate.

5.2 Transparent and Opaque Relaying

WRAP allows the relaying to be *transparent* in the sense that each IRT is simply a sequence of IPv4 addresses designating relay nodes and endpoints, an *Internet Relay Path (IRP)*. The IRT can also be *opaque* so that a holder of the IRT cannot determine the relay path nor can it forge a valid IRT.

Using a transparent IRT, the relay is stateless in the sense that the relaying only relies on routing/directory state and configuration state; it does not require state to be created on name lookups. In this mode of

operation, the relay node is statically configured with an IPv4 address for each of the other realms it connects to, so that an address uniquely identifies which direction to relay a packet (and a particular "egress" address in that realm.) Upon receipt of a WRAP packet, the relay replaces the IP destination with the first forward token component, uses the egress address as the new IP source, and places the old IP source as the last component in the reverse token, as described in the earlier example. This is the simplest possible relaying action, requiring only 4 words in the packet headers to be modified.

To make the WRAP addressing opaque to an observer, the relay node can choose to put a random value in the IRT and translate it to/from IPv4 addresses using the relay table described earlier. This opaque form prevents an upstream source from fabricating IRTs, forcing it to rely instead on the directory service to supply IRTs. In particular, an ISP can retain control of routing, preventing customers from using unauthorized routes. It also prevents a third-party observer from determining protected information from addresses in the packets.

An IRT must have the *reversibility property*, namely that the component-wise reversal of the received IRT provides an IRT that can be used to send a packet back to the source of the packet using the relay from which the original packet was received.

An IRT normally also has the *concatenation property*, i.e. if the IRT to a relay is X from a host H and Y is the IRT to a destination D relative to this relay, then XY is an IRT to destination D from host H . The directory indicates whether the returned IRT supports concatenation or not.

5.3 Multicast

WRAP supports the EXPRESS [4] single-source model of multicast. Multi-source multicast applications can be supported by relaying the multicast through a node that is a source of an existing relay multicast channel, similar to the rendezvous point in PIM-SM, but performed at the WRAP or the application layer.

A subscriber joins a multicast channel by specifying its name in a DRP lookup, which returns the required multicast channel addressing information.

A multicast WRAP header contains the same multicast address G repeated r times, where r represents the maximum number of relay hops in the multicast tree; this allows multicast WRAP relaying to be performed identically to unicast WRAP relaying. As multicast packets are relayed, group addresses may be translated so that the (S,G) pair upon which IPv4 routers do multicast delivery is unique. However, a single intra-realm channel can be reused within a realm to deliver multiple inter-realm channels.

5.4 Content Routing and WRAP

WRAP allows the directory to dictate a path for packets to take from the client to receive the delivery performance it has determined, rather than leaving the client packets to be routed by a separate mechanism, as occurs now.

WRAP also allows the directory to return a path specific to a requesting client, rather than an address that is generally common for all clients.

Finally, WRAP allows a server behind a NAT box to be addressed without creating translation state in intermediate nodes.

5.5 Secure Communication

TRIAD includes a secure communication facility similar to IPsec, i.e. end-to-end at the (inter)network layer. It differs primarily in working in the presence of NAT or WRAP translation, because the Integrity Check Value (ICV) does not include the packet addressing information, similar to the TRIAD-TCP pseudo header. Here, the principal associated with the connection can be identified by name.

6 Additional TRIAD Benefits

TRIAD provides benefits in mobility, VPNs, policy-based routing and extended reverse path forwarding checking, in addition to its support for content routing, NAT and scalable addressing, as outlined below.

6.1 Mobility

For mobile operation in TRIAD, a host visiting a guest network receives a temporary visitor name in that network (in a DNS domain of the visited network) which allows it to then communicate with the rest of the Internet. If the host needs to be reachable or authenticated as its normal DNS name, it gets its home directory service to insert a redirect this name to its current temporary visitor name in the guest network. It

also notifies the guest network of its home identity, based on name. When another host attempts to contact the visitor by its normal name, the home directory provides a redirect to the temporary visitor name, causing the other host to then contact the mobile host directly in the guest network.

When the mobile host moves, transport connections can continue to function even though its address may change. The mobile host simply acquires a guest name in the new network and registers its real name with the guest network and its guest name with its home network. The transport connections simply rebind based on the name identification, the same as required when NAT translation state was lost or changed in the network. For real-time hand-off between guest networks, the mobile host can request that a relay node in the old network forward its packets (using encapsulation) to the new guest network for some limited period of time. This forwarding is canceled before the relay node reuses the address and name that was used by this guest host (allowing the relay node to use common state and time-out mechanisms to control the forwarding and the reuse). A reverse name lookup can also return the "real" name that redirects to this (temporary) name, providing what might be called *reverse aliasing*. A lookup on this real name is used to validate this reverse alias.

With this approach, the key mechanism to support mobility is the adding and removing of redirects in the home directory of the mobile host and the registering of the mobile host in the guest realm. The guest network simply needs to allocate and reclaim temporary addresses and names the same as supported by current DHCP services. It does not require routing all packets to a mobile host through the home gateway for the mobile host or encapsulating traffic to and from the mobile host, as mobile IP proposals imply.

6.2 Virtual Private Networks (VPN)

Using WRAP, an ISP can provide a *Virtual Private Network (VPN)* service by creating for each enterprise a secure (virtual) transit realm that connects to each of its enterprise sites. The enterprise directory and routing system only needs to deal with the topology of the enterprise network with this "virtual" realm directly interconnecting all the sites. The different sites of the VPN can even have overlapping IPv4 address assignments (typically 10.X.X.X) yet still communicate directly without renumbering.

The ISP implements this virtual realm simply by providing routing and secure communication between each site. In this case, the overhead from WRAP is 12 bytes. As an optimization, the ISP can provide at each site a relay node address for each other remote site in the VPN so packets can be addressed as though each site was directly connected through a relay to each other site, reducing the header overhead to 8 bytes. Thus, WRAP can also obviate the need to deploy MPLS [14].

6.3 Policy-based Routing

WRAP supports policy-based routing across multiple realms⁶ by using the WRAP path-based addressing to direct packets through certain relay nodes and to avoid others, with the directory mapping particular names to these policies. For instance, a special name in the name lookup can provide a special IRT to a destination that directs packets over a more secure ISP network to a particular destination rather than using a cheaper but less secure route. The ISP directory service can also provide different IRTs based on the class of service that the requesting customer is paying for. It is similar in this sense to source routing and tunneling, but with the key differences discussed in Section 10.

This routing control can also be used for traffic engineering.

6.4 Extended Forwarding Path Check

WRAP supports an *extended forwarding path (EFP)* check based on the WRAP header indicating the (relay) path it took to the receiver, not just the port that the packet arrives on. The receiver can verify that the packet was received from trusted relay node based on the IP source address, only trusting the local network realm to prohibit source spoofing. It can further rely on the relay node to only accept packets from trusted relay nodes in other realms. With this constraints, a reverse path name lookup reliably yields the name of the source, at least to within some originating domain.

With this approach, the true source of the packet is explicitly specified in the packet, up to the trustworthiness of the relay nodes. The conventional reverse path forwarding check is only used within a local realm to prevent local source spoofing. Thus, a receiver or relay can check whether the relays that the packet took

⁶Each realm can support its own local policy-based routing.

are trusted and accepted, independent of whether it would forward a packet to the source of this packet back along the same path.

Unlike conventional source routing, WRAP operates with strict reverse path forwarding (RPF) checking in place and does not allow source spoofing attacks.

RPF checks at the IP and WRAP levels are important because so-called source spoofing is the basis for many denial of service and security attacks. These attacks and various forms of network device failures and misconfiguration are a growing concern with the scaling of the Internet. A key part of handling these problems is having a reliable means of verifying the true packet source. Encryption techniques providing authentication and confidentiality can, by their cost in processing, actually make denial-of-service a bigger problem. That is, the increased time to decrypt a packet with secure communication, only to discover it is a bogus packet, means a node loses more resources in an encrypted denial-of-service attack than with plaintext messages. (Providing wire-speed hardware-supported encryption addresses this problem in part, but is an expensive solution for low-end systems and generally does not deal with setup processing, such as PKE-based authentication on connection setup.) For mission-critical applications, denial-of-service may be as damaging an attack as any of the other possible security attacks. We view ERPF allowed by WRAP as an important feature for scaling anti-source spoofing and dealing with these DoS concerns.

7 Implementation and Evaluation

We have developed a prototype implementation of the extended directory and routing service required in TRIAD. The key issues are the client name lookup performance and the directory/routing storage and maintenance overhead.

Regarding name lookup, we expect most environments to use transparent IRTs, which have the *concatenation property* mentioned in Section 5. Thus, the caching of names behaves the same as with current DNS because a name server can lookup the address to a server once, then send name requests using the relay fast path to this server rather than through the name service on each intervening relay node. The concatenation property also allows addresses looked up for one client to be used for another client on the same "side" of a relay node. Caching of names thus behaves the same as with current DNS.

Furthermore, content lookups would be typically handled by a content cache on the path to a primary content server, providing faster client service than the current Internet and keeping the name lookup (or content routing) local to this portion of the Internet close to the client.

Opaquing the IRTs can defeat caching, particularly if the returned IRT encodes source dependencies, but the cost is low compared to the other overheads with secure connection setup.

On a name cache miss, in TRIAD, the name lookup may proceed through several relay nodes, causing a full name lookup at each relay node. In contrast, a conventional DNS name cache miss (within an enterprise) causes a DNS request to be sent to a root name server. Thus, TRIAD may use more cycles in total, summed across several relay nodes, but it distributes this load over the relay nodes on the path of communication. In contrast, DNS incurs fewer total lookup cycles but concentrates the demand on the smaller number of root servers.

The number of name suffixes which must be searched is large, but not unacceptably so. There are currently 1.7 million second-level names in use world-wide, e.g. Harvard.EDU, Ietf.ORG, etc. (This number closely matches the number of suffixes obtained from the experiment explained below.) Assuming 64 bytes of space per entry (including hash indexes, etc.), storing the whole name database would cost 128 megabytes, an insignificant amount of disk space, even if the number was to be 10 times as much by the time TRIAD was deployed. NBRP table lookup is not on the packet forwarding fast path, unlike IP routing, so time spent searching the table is typically only paid during connection setup rather than per-packet. Note also that a name lookup already encounters the cost of searching through a database of this size in conventional DNS.

In sum, we expect TRIAD name lookup to have comparable performance and scaling as current DNS, differing primarily for portions of the Internet configured for greater security requirements than supported by current DNS.

Considering the directory and routing overhead, at the ISP level, the name aggregation generally closely matches the address and routing aggregation. For example, Harvard.EDU corresponds to a small number of IP address ranges that further correspond to a small number of routes. This strong correspondence means the aggregation feasible with routing table entries is largely intact in going to name-based routing and

directory services. Conversely, organizations with large numbers of hosts scattered throughout the Internet are uncommon.

7.1 Expected NBRP Performance

To evaluate the expected performance of name-based routing in the current Internet, we processed a comprehensive list of address-to-name mappings in the Domain Name System[19] and BGP table dumps from the MAE-East exchange point[20] by the following algorithm, making the assumption that address realm boundaries roughly correspond to current BGP autonomous system boundaries:

1. Each address range from the BGP table is matched with the DNS zones represented. (If fewer than *site.threshold* hosts in a range belong to an existing zone, they are removed from the table completely and assumed to be handled with the redirection mechanism.)
2. Names whose associated routing information is made redundant by a superzone are also removed.
3. Aggregates are created for any set of names larger than *aggregate.threshold* that have identical routing information (i.e., all known routes were identical, not just the preferred route.)

The resulting aggregates match those expected to be generated in a TRIAD relay node.

One representative set of results is shown in Table 1. These numbers indicate that NBRP results in a

<i>site.threshold</i>	Affixes (1000s)	<i>aggregate.threshold</i>			
		3	5	10	20
2	1727	19.5 (6.7)	20.1 (5.6)	25.7 (4.4)	37.0 (3.4)
3	1692	14.9 (5.9)	16.1 (5.0)	20.6 (4.0)	30.1 (3.2)
10	1679	14.8 (5.9)	16.0 (5.0)	20.6 (4.0)	30.3 (3.2)
original BGP	68.2	11.8			

Table 1: Number of routes (and aggregates) in thousands for different site and aggregate threshold values. With a site threshold of 10 and an aggregate threshold of 3, NBRP produces approximately 14,800 routing table entries (and 5,900 aggregates) which improves significantly on the original BGP number of 68,200 routing table entries.

set of destinations (and thus update frequency) comparable to BGP; higher-level aggregation may be able to reduce this yet further without resorting to renumbering or renaming.

BGP does have a limited mechanism for aggregation: a single route update may include several address prefixes. It is not clear the extent to which BGP software makes use of this to optimize update calculations: there is no requirement that advertisements keep these address prefixes together, and the address ranges must appear separately in the IP routing table. The entry in Table 1 corresponding to "original BGP" with an aggregation threshold of 3 indicating 11,800 entries indicates the best possible number of routes with BGP aggregation.

Addition of a new name is common, unlike addition of new BGP prefixes, and this name information must propagate to all relay nodes. However, addition of new names is done on human time scales; during the recent past, third-level domain names have been added at about 12 per minute. To put this in perspective, a backbone router may receive more than 2,000 routing updates per minute. Also, the actual level of routing updates necessary for new names is lower because changes to aggregates can be "batched" to reflect many new names with one update.

7.2 WRAP Implementation and Performance

WRAP incurs a low space and time overhead for communication on average because communication within a realm just uses the conventional IPv4 header. Given that most communication is local and the current Internet with NAT boxes is effectively at most 3 relays to anywhere, the packet header overhead on average is expected to be significantly less with WRAP than with IPv6⁷.

⁷One could argue that the Internet does not actually need more global addresses, by relying on efficient allocation and NAPT, given only about 1 percent of the IPv4 addresses are actually in use. However, WRAP is still beneficial for other reasons, such as connecting private address domains, VPNs and content routing.

This header overhead is significant because most packets are small and per-packet processing is a significant cost with small packets. This optimized local communication also suits small embedded systems, many of which use or will use limited bandwidth wireless communication. Moreover, it is readily hardware implementable because of the size of relay address to lookup can be fixed-size.

In comparison to conventional forwarding, relaying requires an additional lookup of the next forwarding component in the context of the virtual interface to which the packet is mapped with the (SA,DA) lookup. A hardware implementation may add an additional lookup resource to handle this or simply perform two lookups on the same memory, depending on the speed of this memory and the forwarding performance requirements. We expect initial hardware implementations may restrict the *offset* they support, throwing packets with larger values to software.

Multicast relaying uses additional state in the forwarding path but is the same implementation and performance.

Our Linux implementation of WRAP added about 1,500 lines of code as a kernel module and incurred an extra 2.2 microsecond overhead (or 2.6 percent) for relaying compared to conventional IP forwarding.⁸ Thus, the complexity is minimal for either software or hardware, and the software performance overhead is minimal, and comparable to that required for NAT forwarding.

8 WRAPID Gateways

A WRAPID (WRAP-to-IP-Domain) gateway allows existing IPv4 end hosts to operate with TRIAD without modifications to their software. WRAPID provides translation between IPv4 addresses and WRAP addressing similar to the IPv4 to IPv4 translation provided by NAT boxes.

A WRAPID gateway handles outgoing conventional DNS lookups, performing a DRP lookup using just the DNS name. The WRAPID gateway allocates an IPv4 address for this remote server and sets up a mapping to the appropriate WRAP header. This header may map directly to the remote host or to a WRAPID gateway that serves that host. When an (IPv4) packet is sent to this allocated address, the WRAPID gateway translates the packet to a WRAP packet with the appropriate header and forwards it onwards. On receiving a WRAP packet from an external host, the WRAPID gateway translates the packet to a simple IPv4 packet with the IP source appearing as this locally allocated address.

The WRAPID gateway also handles incoming DRP requests, performing the name lookup internally, and then requesting a connection setup at the host, using the URL information in the DRP request and sends an HTTP request to the client, if that information is present in the DRP request. It then returns this connection information and splices the client connection into the connection it has established to the server.

The WRAPID gateway can also implement WRAPsec, providing secure communication to the other WRAP endpoint, either a WRAP-enabled host or another WRAPID gateway.

The WRAPID gateway allows WRAP to be deployed incrementally. In particular, one can have hosts on the same subnet being WRAP-enabled while others are not, yet still able to communicate with each other as well as hosts in other address realms. The optimization of eliminating the WRAP header when communicating within the same address realm means that a WRAP-enabled host never sends WRAP packets to other hosts in the same realm, so there is no need to discriminate between these hosts as part of local communication. Only the directory service interfacing to the rest of the Internet needs to distinguish. However, a full TRIAD implementation (with the attendant host changes) is required to provide end-to-end security and reliability.

9 TRIAD Deployment

TRIAD has a simple deployment path, based on user need, allowing TRIAD to be realized as an incremental evolution of the current Internet.

At the content layer, DRP and NBRP can be implemented in firewalls and inter-realm routers with the additional capability to fail over to using the current Domain Name System, etc. to implement the functionality in regions of the network that do not support TRIAD. Similarly, content resolvers can make use of the existing naming infrastructure to locate other TRIAD gateways rather than participating in a dynamic routing protocol. We expect deployment to occur mainly at the edges of the network, and thus cannot depend on ISPs providing new infrastructure. Such a scenario could lead to a topology that would

⁸The test machine was a 333 MHz Celeron with 128 MB of RAM, running Linux 2.2.13.

have many thousands of realms peering in the "global" Internet, but there is little need for NBRP in such an environment. The amount of topological change in such a situation is small, and multihomed sites can easily list all their gateways as NS records rather than constantly updating the DNS. Later, these CRRs can be upgraded to content routers as ISPs begin offering NBRP.

Deploying TRIAD for NAT is also compelling. Consider as an extreme example a foreign country with limited IPv4 addresses such as Thailand. The limitations of conventional NAT make it questionable as a solution to providing more addresses yet moving to IPv6 does not make sense either, given the limited deployment of IPv6, the limited product support, and the need to communicate with the IPv4 portion of the Internet. However, with TRIAD, each such country can install a WRAP relay router that interfaces to the Internet. Attached to this top-level relay are one or more WRAPID gateways that include conventional NAT capability. The conventional NAT capability allows these hosts to communicate with the existing conventional IPv4 Internet. Each ISP, country or even organization that adopts TRIAD is able to communicate with other organizations using TRIAD *without* consuming any of its global IPv4 addresses⁹. For instance, if Thailand and Indonesia both adopt TRIAD, they then have virtually unlimited addresses internally and between themselves, and are only constrained on the number of addresses they have available to communicate with the current Internet. (This is actually the same situation as if they had internally converted to IPv6, given they would still have to communicate with the rest of the planet using IPv4. But, with IPv6, they would also have to upgrade all their existing hosts and networking infrastructure.)

Thus, each organization is motivated to adopt TRIAD because it allows them to communicate with other TRIAD organizations without using their limited global IPv4 addresses, and because it makes it easier for other TRIAD users to communicate with them. So, those organizations that are currently short of addresses are motivated to move to TRIAD and those that are not are still motivated if they are interested in having the former communicate with them. Given that most of the major web sites are in the United States, and the U.S. companies have been in the lead to build Web-based operations, there would be considerable commercial motivation to support TRIAD in the American web sites once foreign companies were using TRIAD among themselves.

This initial deployment requires no real changes to end hosts and no change to the basic IPv4 routers and switches constituting the infrastructure of the leaf and backbone networks. It only requires the deployment of content routers and WRAPID gateways, but these are modest extensions of the current NAT-enabled routers. Here, we assume that end-user applications have been or will be modified in any case to deal with the lack of meaning of addresses across NAT boundaries.

Once WRAP is deployed to some degree in the Internet, first host implementations are expected to arise with large-scale servers where eliminating the extra overhead, delay and point of failure of a WRAPID gateway may be warranted. Making an externally accessed server WRAP-enabled also eliminates the server use of an externally visible (IPv4) address which, with an active server, would be essentially allocated indefinitely to this server. During this transition, conventional IPv4 hosts and TRIAD-aware hosts can easily and efficiently co-exist in the same address realm. Given that WRAP appears relatively straight forward to implement, the main delay in getting all hosts upgraded to WRAP is expected to be the basic inertia in getting changes into commercial software and getting administrators of systems to upgrade their software. Hosts that need end-to-end security and reliability are also motivated to upgrade to native WRAP.

Consequently, TRIAD is readily deployable incrementally. There is no need to change the network infrastructure within an address realm or to change backbone routers and management. The boundary (NAT) routers are upgraded to support TRIAD and then the hosts can then be individually upgraded to use WRAP natively.

10 Related Work

The original Internet directory service was supplied by a "hosts.txt" file that listed all hosts in the Internet. As the Internet grew, this approach was replaced by DNS [6] in 1985. Subsequent work on so-called network directories such as X.500 have suffered from misguided objectives of supporting naming of other types of objects such as mailboxes and providing more flexible ways of specifying identification, such as lists of attributes.

⁹This assumes the gateway already has one such address if the WRAP relays communicate over the existing wide-area IPv4 infrastructure.

TRIAD draws on the direction established in the web of treating both host name and file name as part of the path name to content, and unifies the handling of these two portions of the URL. It further builds on the *decentralized naming* approach advocated from experience in the V distributed system [21], and effectively implemented in file systems, which can be summarized as: Names are external identification of objects and a server names what the objects it implements — nothing more and nothing less”.

Current wide-area content routing depends on HTTP or DNS-level redirect. For instance, Cisco's Distributed Director (DD) redirects a name lookup from the main site to a replica site closer to requesting client address, based on responses from a set of participating routers running an agent protocol, supporting DD. Unfortunately, the client incurs the response time penalty of accessing this main site DD before being directed to the closer site. Proprietary schemes by Akamai, Sightpath, Arrowpoint and others appear to work similarly.

Web caching was introduced by the Harvest project, spawning a whole industry of vendors. Transparent caching evolved to eliminate the need for explicit configuration and the difficulty of configuring hierarchical caches.

Protocols such as ICAP, Cisco's WCCP and Arrowpoint/Cisco's CAPP define communication between web caches, web caches and routers, and other content distribution devices. To date, these and other proprietary protocols have not be architected into a coherent Internet architecture.

The XNS [23] Sequenced Packet Protocol (SPP) used a separate connection setup protocol, similar to the separation between DRP and TRIAD TCP we are proposing.

NAT was introduced into the Internet in Jacobson's insightful early proposal [11] although the same techniques appear in some earlier distributed systems work [10]. Since 1992, various RFC's [12] have clarified the use of NAT, provided for private addresses [13] and clarified the terminology, use and problems [17]. Industry has deployed a variety of products supporting network address translation, including firewalls, routers and server load balancing switches. More recently, work on IPsec and others have recognized the problems with basing identity on IP addresses and the conflict of end-to-end security with the increasing deployment of NAT.

RSIP [15] is an approach to dealing with NAT, where a host in a NAT realm explicitly obtains an external IP address, tunnels packets through the NAT gateway using this external IP address and thus can use IPsec and other protocols without requiring NAT translation. However, RSIP requires host modification to operate in this mode and it does not increase the number of external IP addresses. With all the extra benefits that TRIAD provides, it seems more effective and lower risk to modify the hosts to support native TRIAD.

The path-based WRAP relay model of addressing as an extension of the basic forwarding level of conventional routers is similar in some respects to the Sirpent [1] form of loose source routing except WRAP is designed to work with IPv4. WRAP relaying is similar to loose source routing except the packet is forwarded at each relay with the source IP address that of the relay, not the original source address. Moreover, each specified address may be in a separate address realm, with translation between address realms occurring at each realm boundary. Source routing provides a source-controlled path but does not cross realms and does not change the source address on each hop, as is required for inter-realm communication. TRIAD path-based addressing could be provided as a new IP-level option. However, using a separate shim header seems preferable because of the inefficiency of routers handling packets with IP options, given that some options need to be handled by each router and others only need to be handled by the IP-addressed endpoint. Moreover, WRAP makes it easier for a hardware implementation to determine the offset of the transport header, which is important for layer 4 access control lists. WRAP is similar in this respect to IPv6 header extensions.

IP tunneling has been used to effectively extend addressing by tunneling from one realm to another. However, tunneling makes layer 4 filtering harder because, with multi-hop tunneling, the location of the layer 4 header involves parsing each encapsulation. Also, unlike WRAP, the path the packet takes is lost with tunneling. Moreover, tunneling incurs greater overhead than WRAP and requires that the source know the path. Moreover, the packet size does not change with WRAP, unlike encapsulation and de-encapsulation that occurs with tunneling.

MPLS [14] provides tagging of packets similar to WRAP, but below the IP level. MPLS does not provide more addresses beyond that provided by NAT, unlike WRAP. On the other hand, WRAP can be used intra-

realm and inter-realm for traffic engineering and VPNs, reducing, if not eliminating, the need for MPLS¹⁰. MPLS also requires special support in the forwarding path of *all* routers on the path, whereas WRAP and TRIAD only require support at the border or relay nodes. MPLS also requires a new mechanism for distributing tags. MPLS does not save the path a packet followed either. While the WRAP header does impose a higher overhead than an MPLS tag, it is less than IPv6 and less than conventional IPv4 tunneling, especially with multi-path tunnels. Thus, IP4 plus MPLS is not a solution to scaling and IPv6 plus MPLS carries all the disadvantages of both. Both WRAP and MPLS make the offset of the TCP/UDP ports variable within the packet, affecting the design of access control filters on packets. However, with the length field in the WRAP header at a fixed offset, it is straightforward for even a hardware implementation to determine the actual offset of layer 4 ports, as required for access control processing. Moreover, in initial deployment, we expect that firewalls may simply restrict WRAP packets to specific WRAP-enabled hosts, such as WRAPID gateways, which can filter further as needed.

Recent IETF work has promoted "transparency" as an important property to achieve in the Internet, defined as "a single universal logical addressing scheme and the mechanisms by which packets may flow from source to destination essentially unaltered" [16]. We view that TRIAD provides transparency under this definition, viewing the "logical addressing scheme" to be DNS naming and the transmission of data without changing the data or its checksum as "essentially unaltered". The changing of the addressing in the packet is not real alteration because corruption by intermediate points is as detectable as with conventional end-to-end delivery.

The restriction of IP multicast to single-source was proposed in EXPRESS [4]. This single-source multicast approach is now being deployed.

11 Concluding Remarks

TRIAD is a promising candidate for the next generation Internet architecture. It addresses the key problem of scaling content distribution by defining a *content layer* that directly supports efficient content routing, transparent caching and content transformation. It further supports network address translation while providing end-to-end semantics, reliability and extensible addressing. Besides these benefits, TRIAD also provides a significantly improved directory service as well as innovative approaches to mobility, virtual private networks, policy-based routing and source spoofing. Compared to IPv6, TRIAD is more backwards compatible, more deployable, more efficient and more secure while providing the same end-to-end semantics and recovery relative to network failures.

TRIAD, as the name suggests, is based on three key ideas. First, TRIAD places the external character-string name as the means of identification of endpoints, relegating the packet address to the role of a transient routing tag. In doing so, it makes network naming consistent with that used in file systems, where similar hierarchical names map to files, and internal system identifiers or handles are generated and used for efficiently in the file access operations. In fact, one can recognize both network and file-level names designate content or state, and see their combined usage in web URLs.

Second, TRIAD integrates naming, routing and connection setup into a content layer, recognizing name lookup needs routing information to locate the closest replica to the requesting client. It also needs the same reliability, security and performance as routing. The integration of transport-layer connection setup with the name lookup allows the name lookup to be end-to-end while reducing the roundtrip delays for content access. As the bandwidth of the Internet increases to multi-gigabit rates, roundtrip times are becoming the dominant client performance issue. This integration also facilitates better failure handling between the directory and transport layer.

Finally, TRIAD extends packet addressing with the ability to specify a variable-length path to the destination in a shim protocol called WRAP, allowing the directory service to control the path a packet takes. This path addressing also provides extensible addressability between address realms, efficient virtual private networking and scalable anti-source spoofing. The simplicity of WRAP makes it feasible to implement in hardware in the next generation of switch/routers, allowing wire speed relaying, even at the highest performance levels. Moreover, intra-realm communication can optimize out WRAP, incurring the same packet overhead in size and processing as IPv4.

¹⁰One of the original motivations for MPLS, efficient IP forwarding, has been eliminated by the advent of wire-speed hardware IPv4 forwarding engines.

Our current work is focused on designing and implementing the detailed protocols required by TRIAD and performing further evaluation and study to support this direction. Specifically, we are performing much more comprehensive studies of the large-scale behavior of name-based TCP and routing through simulation. The ability of name rebinding to handle routing topology changes and the effects of route aggregation need to be clearly demonstrated before TRIAD can achieve wide-scale deployment. However, we have confidence in TRIAD's scalability, since the dynamics of naming and routing are similar to what already exists in the IPv4 Internet.

We believe the primary competition to TRIAD at this stage is the continued *ad hoc* deployment short-term fixes and specialized mechanisms, including the proprietary approaches that have arisen for dealing with content distribution. Continued growth of the Internet without a guiding architecture risks increasing entropy as a result of these fixes, detracting overall from its future reliability, availability and security. We see TRIAD as an alternative to this unfortunate direction.

References

- [1] David R. Cheriton, Sirpent, Proc. SigComm'89, 1989.
- [2] David R. Cheriton and Chetan Rai, Wide-area Relay Addressing Protocol (WRAP), in preparation. 1999.
- [3] Mark Gritter and David Cheriton, Name-based Routing Protocol Specification, in progress, 1999
- [4] Hugh Holbrook and David R. Cheriton, IP Multicast Channels: EXPRESS Support for Large-scale Single-source Applications, Proc. ACM SigComm'99 Cambridge, MA Sept. 1999.
- [5] Anonymous (to allow for blind review), Name-enabled Routing and Directory Services in TRIAD, in progress, 2000.
- [6] P. Mockapetris, Domain Names - Concepts and Facilities, RFC 882, November, 1983 (obsoleted by RFC 1034 and 1035).
- [7] D. Eastlake, Domain Name Security Extensions, RFC 2535, March 1999.
- [8] P. Ferguson, H. Berkowitz, Renumbering: What is it and why do I want it anyway, RFC 2071, January 1997.
- [9] S. Deering and R. Hinden, IP Version 6 Addressing Architecture, RFC 2372, July 1998.
- [10] D.R. Cheriton, Local Networking and Internetworking in the V-System, Proc. 8th Data Communication Symposium, IEEE/ACM, 1983
- [11] V. Jacobson, LNAT — Large-scale IP via Network Address Translation, working draft, Lawrence Berkeley Labs, Jan. 1992.
- [12] E. Egevang and P. Francis, The IP Network Address Translator (NAT) RFC 1631, May 1994.
- [13] Y. Rekhter, B. Moskowitz, D. Karrenberg and G. de Groot, Address Allocation for Private Internets, RFC 1597, March 1994.
- [14] E. Rosen, A. Viswanathan and R. Callon, Multi-protocol Label Switching Architecture, work-in-progress, Internet draft, 1999.
- [15] M. Borella, D. Grabelsky, J. Lo, Realm Specific IP: Protocol Specification, draft-ietf-nat-rsip-protocol-03.txt, Oct. 1999 (work in progress).
- [16] M. Kaat, Overview of IAB 1999 Network Layer Workshop, draft-ietf-iab-ntwlyrws-over-01.txt, work in progress, Oct. 1999
- [17] NAT Working Group, P. Srisuresh and Matt Holdrege, IP Network Address Translator (NAT) Terminology and Considerations draft-ietf-nat-terminology-00.txt, work in progress, July 1998

- [18] S. Kent, C. Lynn, and K. Seo, "Secure Border Gateway Protocol (S-BGP)", to appear in IEEE Journal on Selected Areas in Communication, 1999.
- [19] Internet Domain Survey, July 1999, <http://www.isc.org/ds/>.
- [20] Internet Performance Measurement and Analysis project, <http://www.merit.edu/ipma/>.
- [21] D. R. Cheriton and T.P. Mann, Decentralizing a Global Naming Service for Improved Performance and Fault Tolerance. ACM TOCS, May 1989
- [22] Wireless Application Protocol Forum <http://www.wapforum.org>.
- [23] Xerox Network Services Specification, Xerox Corporation, 1980.

Distributed Packet Rewriting and its Application to Scalable Server Architectures*

Azer Bestavros Mark Crovella Jun Liu
Computer Science Department
Boston University
Boston, MA 02215
{best,crovella,junliu}@cs.bu.edu

David Martin†
Dept of Math and CS
University of Denver
Denver, CO 80208
dm@cs.du.edu

Abstract

To construct high performance Web servers, system builders are increasingly turning to distributed designs. An important challenge that arises in such designs is the need to direct incoming connections to individual hosts. Previous methods for connection routing (Layer 4 Switching) have employed a centralized node to handle all incoming requests. In contrast, we propose a distributed approach, called Distributed Packet Rewriting (DPR), in which all hosts of the distributed system participate in connection routing. DPR promises better scalability and fault-tolerance than the current practice of using centralized, special-purpose connection routers. In this paper, we describe our implementation of four variants of DPR and compare their performance. We show that DPR provides performance comparable to centralized alternatives, measured in terms of throughput and delay. Also, we show that DPR enhances the scalability of Web server clusters by eliminating the performance bottleneck exhibited when centralized connection routing techniques are utilized.

1. Introduction

The phenomenal, continual growth of the World Wide Web (Web) is imposing considerable strain on Internet resources, prompting numerous concerns about the Web's continued viability. In that respect, one of the most common bottlenecks is the performance of Web servers—popular ones in particular. To build high performance Web servers, designers are increasingly turning to distributed systems. In such systems, a collection of hosts work together to serve Web requests. Distributed designs have the potential for scalability and cost-effectiveness; however, a number of challenges must be addressed to make a set of hosts function efficiently as a single server.

Connection Routing: Consider the sequence of events that occur as a result of a client requesting a document from a Web server. First, the client resolves the host's domain name to an initial IP address. Second, the IP address itself may represent a distributed system, and one of the hosts in the system must be chosen to serve the request. There are many ways to perform the first mapping (from domain name to initial IP address). For example, this mapping could be coded in the application as is done within Netscape Navigator to access Netscape's Home Page [8]. Alternately, this mapping could be done through DNS by advertising a number of IP addresses for a single domain name. Similarly, there are many ways to perform the second mapping (from initial IP address to actual host). For example, this mapping could be done at the application level, using the HTTP redirection approach [1] or using a dispatcher at the server [2, 17].

While initial attempts to implement connection routing for scalable Web servers focused on using the mapping from domain names to IP addresses [10], recent attempts have focused on the second kind of mapping (IP addresses to hosts) because of the potential for finer control of load distribution. One common feature of all of these attempts (whether proposed or implemented) is that a centralized mechanism is employed to perform the mapping from IP addresses to hosts. Examples include the Berkeley MagicRouter [2], the Cisco Local Director [17], and IBM's TCP Router [7] and Network Dispatcher [9].

Distributed Connection Routing using DPR: In contrast, DPR is a technique that allows the mapping between IP address and host to be implemented in a *distributed*, efficient, and scalable fashion. In particular, DPR can be viewed as a distributed method of mapping m IP addresses to n servers.¹ Using DPR, every host in a Web server cluster acts *both* as a server and as a connection router. Thus, unlike existing solutions that rely on a single, centralized connection router, DPR enables both the service and the routing responsibilities to be

*This work was partially supported by NSF research grants CCR-9706685 and CCR-9501822.

†Research completed while co-author was at Boston University.

¹If $m = 1$, then DPR becomes similar to the centralized solutions mentioned above—the difference being that DPR allows *both* packet routing and service to be combined on the same node.

shared by all hosts in the cluster. Distributing the connection routing functionality allows for true scalability, since adding a new host to the cluster automatically adds enough capacity to boost *both* Web service and connection routing capacities.

To illustrate the benefits of using DPR, consider the problem of scaling up a Web site that initially consists of a single server host. Adding a second server host using typical existing solutions (for example, Cisco's Local Director [17], or IBM's NetDispatcher [9]) requires using special-purpose hardware to distribute incoming HTTP requests between the two server hosts. This kind of centralized solution provides connection routing capacity that far surpasses what a two-host server is likely to require. In other words, the upgrade path (and hence the price tag) for a centralized solution is not truly incremental: the two-host server will be roughly three times the cost (if an ordinary PC is used as a centralized router) and may reach ten times the cost of a single-host server.

An additional, important issue for many content providers is that the centralized solution creates a single-point-of-failure in the system, which leads to even more costly solutions such as using a second, standby connection router. Thus for mission-critical Web sites, centralized connection routing escalates the imbalance in capacity between connection routing and connection service. These problems disappear when using a DPR-based architecture. Adding a second server to the site requires no special hardware, introduces no single-point-of-failure, and utilizes the added capacity (and hence dollars spent) to scale both the connection routing and connection service capacities equally.

Paper Contribution and Scope: The novelty of DPR lies in its *distribution* of the connection routing protocol (Layer 4 Switching), which allows all hosts in the system to participate in request redirection, thereby eliminating the practice of using a special purpose connection router to achieve that functionality.

DPR is one of the salient features of COMMONWEALTH—an architecture and prototype for scalable Web servers being developed at Boston University. The design of DPR is driven by a large set of goals that the COMMONWEALTH architecture strives to achieve. These goals are:

1. *Transparency:* Clients should not be exposed to design internals. For example, a solution that allows a client to distinguish between the various servers in the cluster—and hence target servers individually—is hard to control.
2. *Scalability:* Increasing the size of the cluster should result in a proportional improvement in performance. In particular, no performance bottlenecks should prevent the design from scaling up.
3. *Efficiency:* The capacity of the cluster as a whole should be as close as possible to the total capacity of its constituent servers. Thus, solutions that impose a large overhead are not desired.
4. *Graceful Degradation:* The failure of a system component should result in a proportional degradation in the

offered quality of service. For example, a solution that allows for a single point of failure may result in major disruptions due to the failure of a miniscule fraction of the system.

5. *Connection Assignment Flexibility:* Connection assignment techniques should be flexible enough to support resource management functionalities—such as admission control and load balancing.

In the remainder of this paper we show how DPR supports these goals in the construction of the COMMONWEALTH server. In the next section we review related work and show why DPR is different from previous proposals for connection routing in Web servers. Then in Section 3 we describe the design tradeoffs for DPR and the variants of DPR that we have implemented and tested in our laboratory. In Section 4 we show performance results using DPR, indicating that DPR induces minimal overhead and that it achieves performance scalability superior to that achievable using existing centralized connection routing. Finally, in Section 5 we conclude with a summary.

2. Related Work

Preliminary work on scalability of Web servers has been performed at NCSA [10] and DEC WRL [14]. In both cases, load is balanced across server hosts by providing a mapping from a single host name to multiple IP addresses. In accordance with DNS standard, the different host IP addresses are advertised in turn [16]. In addition to its violation of the transparency property discussed in the previous section, both the NCSA and DEC WRL studies observe that this “Round Robin DNS” (RR-DNS) approach leads to significant imbalance in load distribution among servers. The main reason is that mappings from host names to IP addresses are cached by DNS servers, and therefore can be accessed by many clients while in the cache. The simulations in [7] suggest that, even if this DNS caching anomaly is resolved, the caching of Host-to-IP translations *at the clients* is enough to introduce significant imbalance.

Rather than delegating to DNS the responsibility of distributing requests to individual servers in a cluster, several research groups have suggested the use of a local “router” to perform this function. For example, the NOW project at Berkeley has developed the MagicRouter [2], which is a packet-filter-based approach [13] to distributing network packets in a cluster. The MagicRouter acts as a switchboard that distributes requests for Web service to the individual nodes in the cluster. To do so requires that packets from a client be forwarded (or “rewritten”) by the MagicRouter to the individual server chosen to service the client's TCP connection. Also, it requires that packets from the server be “rewritten” by the MagicRouter on their way back to the client. This *packet rewriting* mechanism gives the illusion of a “high-performance” Web Server, which in reality consists of a router and a cluster of servers. The emphasis of

the MagicRouter work is on reducing packet processing time through "Fast Packet Interposing", not on the issue of balancing load. Other solutions based on similar architectures include the Local Director by Cisco [17] and the Interactive Network Dispatcher by IBM [9].

An architecture slightly different from that of the MagicRouter is described in [7], in which a "TCP Router" acts as a front-end that forwards requests for Web service to the individual back-end servers of the cluster. Two features of the TCP Router differentiate it from the MagicRouter solution mentioned above. First, rewriting packets from servers to clients is eliminated. To do so requires modifying the server host kernels, which is not needed under the MagicRouter solution. Second, the TCP Router assigns connections to servers based on the state of these servers. This means that the TCP Router must keep track of connection assignments.

The architecture presented in [11] uses a TCP-based switching mechanism to implement a distributed proxy server. The motivation for this work is to address the performance limitations of *client-side* caching proxies by allowing a number of servers to act as a single proxy for clients of an institutional network. The architecture in [11] uses a *centralized* dispatcher (a Depot) to distribute client requests to one of the servers in the cluster representing the proxy. The function of the Depot is similar to that of the MagicRouter. However, due to the caching functionality of the distributed proxy, additional issues are addressed—mostly related to the maintenance of cache consistency among all servers in the cluster.

3. Implementation of DPR

As described in Section 1, our goals in developing DPR were transparency, scalability, efficiency, fault tolerance, and flexibility in connection assignment. Previous centralized approaches (described in Section 2) have focused on transparency and load balance: these are natural features deriving from a design using centralized routing. The two dominant styles of centralized routing are shown in Figure 1 (a) and (b). Figure 1 (a) shows the MagicRouter style, in which packets traveling in both directions are rewritten by a centralized host. Figure 1 (b) shows the TCP router style, in which only packets traveling from the clients are rewritten, still by a centralized host. An important advantage of the TCP router style is that the majority of bytes in a Web server flow from the server to the client, and these packets do not require rewriting.

In contrast to centralized approaches, we seek to address our wider set of goals, which also include scalability and fault tolerance. As a result we adopt a distributed approach to TCP routing, namely distributed packet rewriting. Under DPR, each host in the system provides both Web service and packet routing functions, as shown in Figure 1(c). Under DPR the structure of any connection is conceptually a loop passing through three hosts (client and two server hosts). The entire set may have no hosts in common with another connection on

the same distributed server. We refer to the first server host to which a packet arrives as the *rewriter*, and the second host as the *destination*.

Centralized schemes place the rewriting task within the routers connecting a distributed web server to the internet (or as close to such routers as possible). DPR instead transfers this responsibility to the Web servers it concerns. This can be seen as an instantiation of the end-to-end argument: the choice of the final server is essentially a service-specific decision, and so should be made as close as possible to the service points rather than being distributed throughout general-purpose network components.

Another important advantage of DPR is that the amount of routing bandwidth scales with the size of the system, in contrast to the centralized approaches. Furthermore, since the routing function is distributed, this system can not be wholly disabled by the failure of a single node—as is possible under centralized approaches.

The DPR scheme assumes that requests arrive at the individual hosts of the server. This can occur in a number of ways. The simplest approach (which we currently use) is to distribute requests using Round-Robin DNS. Although requests may well arrive in an unbalanced manner because of the limitations of RR-DNS, hosts experience balanced demands for service because of the redistribution of requests performed by DPR.

Design Tradeoffs

Two design issues arise in determining the specific capabilities of a DPR implementation. First, will routing decisions be based on stateless functions, or will it require per-connection state? Second, how should rewritten traffic be carried on the server network? The following sections investigate these questions and describe decisions made in our various implementations extending the Linux 2.0.30 kernel with DPR support.

Stateless vs Stateful Routing:

It is possible to balance load across server hosts using a stateless routing function, *e.g.*, a function that computes a hash value based on the source and destination IP and TCP port addresses of the original packet. On the other hand, more sophisticated load balancing policies may require more information than what is contained in the packets, for example, knowledge of load on other hosts. In this case, each rewriting host must maintain a routing table with an entry for each connection that is currently being handled by that host.

Stateless Approach: In the stateless approach, we use a simple hash function on the client's IP address and port number to determine the destination of each packet. Since the client's IP/port forms a unique key for requests arriving at the server, this function is sufficient to distribute requests.

Using server logs from the BU Web site in simple simulations, we have verified that our hash function is effective

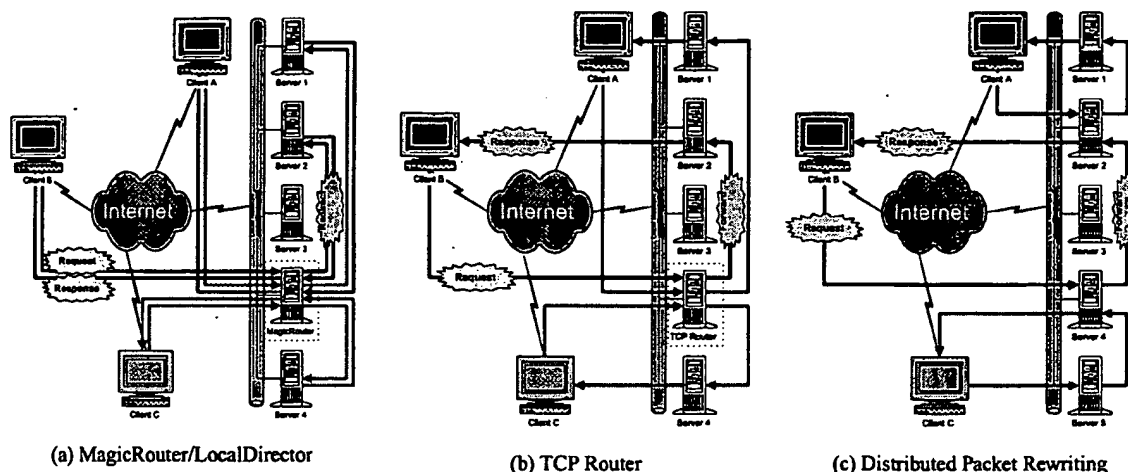


Figure 1. Illustration of various architectures for distributed Web Servers

at balancing load (in terms of hits per server over time) for actual client request arrivals. An important factor in this success is the use of the client port number as an input to the hash function; the client's TCP layer indirectly chooses the final server when it selects its ephemeral TCP port. Successive port numbers from the same client should map to different server hosts, dispersing each burst across servers, and thus alleviating the imbalance due to the burstiness of client requests [7].

Although stateless implementations are lightweight and the resulting server load distributions are acceptable, we must keep in mind the inability of the rewriter to route a connection based on other factors (such as end-server load, distance, availability, or the necessity that successive requests be routed to the same host for correct session semantics).

Stateless/LAN Implementation: This variant takes the simplicity and speed of stateless MAC address rewriting to an extreme. Because no state is stored, the additional code and data required is small. The Stateless/LAN implementation simply overwrites the MAC address of the packet and retransmits it on the LAN. The simplicity of the transformation allows rewriting to occur in the context of the network device driver, namely, in the kernel routine that device drivers use to register incoming packets. This implementation thus receives, rewrites, and retransmits packets all within a single interrupt service cycle; furthermore, no device-specific modifications are required.

While rewriting the entire request in the interrupt routine provides performance virtually indistinguishable from that of a dedicated rewriting router (see Section 4), our Stateless/LAN implementation is not practical. When Stateless/LAN processes fragmented packets, only the first IP fragment contains the necessary TCP port information to ensure proper delivery and subsequent fragments are misrouted.

Because of this shortcoming, we used this implementation only as an indication of an upper bound on the performance that can be achieved with DPR-style techniques.

Stateful Approach: In the stateful approach, the packet routing decision is based on more information than is contained in the packet. For example, a stateful approach is necessary in order to route connections based on the current load on each server host. Most of our efforts have concentrated on this approach.

Stateful Implementation: In the stateful method, rewriters must track TCP connection establishment and termination. A table of connections currently being rewritten is maintained by each host and is consulted in order to rewrite each packet. In implementing these functions we were able to adapt features from code already present in the Linux kernel that supports *IP Masquerading*. IP Masquerading [12] was developed to allow multiple hosts to function behind a firewall without valid IP addresses. Thus, IP Masquerading supports connections that are initiated *from* the "hidden" hosts. In order to support a distributed server, we need to support connections connecting *to* the hidden hosts.

Using the IP Masquerading functions adapted to support a distributed server, the rewriter has considerable freedom to choose a destination when it receives the first packet of a client's TCP stream. After noting its decision in a state table, it then forwards each packet associated with the connection using either the MAC rewriting or IPIP encapsulation technique, depending on the network location of the destination.

At present, the routing decision for a newly observed connection is made by simply obtaining the next entry in a ring of server addresses. This ring is extended to user space through a `setsockopt(2)` system call. By populating the ring intelligently, a user daemon can adjust the rewriting policy as server conditions change.

We note that independently and at approximately the same time as our work, Clarke developed a general-purpose TCP forwarding kernel extension based on IP Masquerading [5] which can also be used to support implementation of distributed Web servers.

Addressing Techniques:

There are two approaches to addressing packets bound for another host in a multiple-server environment, depending on whether the original destination IP address must be communicated from the rewriter to the destination host.

The first approach is appropriate when there is only one published IP address for the whole Web server cluster (as would be the case when a centralized connection router is used). In this case, the original packet's destination IP address (IP_1) is replaced with that of the new destination (IP_2) and then the packet is routed normally. When the new destination transmits packets to the client, it must be careful to replace its IP source address (IP_2) with that of the rewriter (IP_1), because the client believes its connection to be with the rewriter (*i.e.* IP_1). Every host in the Web server cluster knows that its outbound traffic should always bear the source address IP_1 , so the address IP_1 need not explicitly appear in rewritten packets.

One consequence of this technique is that the IP and TCP checksums need to be adjusted when rewriting, since they both depend on the destination IP addresses in the packet. (In practical terms, only the IP checksum is important, since IP routers do not examine the payload of IP packets they encounter [3]. However, firewalls and other types of "smart routers" might in fact examine the TCP checksum, so it is advisable to recompute it as well.)

The second approach applies to systems with more than one published IP address, as in DPR. In a DPR system, a mechanism is needed to communicate both the original address (IP_1) and the rewritten address (IP_2) in packets sent between the rewriter and the destination hosts so that the destination knows how to populate the IP source address field. The most efficient method we used was to rewrite the MAC address of the packet and retransmit, leaving the original packet's IP addresses and checksums undisturbed. (This is how Internet hosts normally send packets through a gateway.) Although fast, the method only works if both servers are located on the same LAN. If the servers are on different LANs, then IP-level routing is necessary. In this case we tunnel the original packet to IP_2 with IPIP encapsulation as described in RFC2003[15]. When the packet arrives at IP_2 , the outer IPIP header is discarded and the inner header is interpreted.

Whether MAC rewriting or IPIP encapsulation is used, the server with primary address IP_2 eventually receives and processes an IP packet bearing the original destination address IP_1 . Therefore, each server must be configured to respond to all of the possible original destination addresses (such as IP_1) in addition to its own primary address. In our Linux implementation, this was just a matter of adding loop-

back alias devices with the extra addresses.

4. Performance Evaluation

In this section we describe the performance of DPR variants. We have two goals: first, to characterize the overheads present in DPR; and second, to study the scalability of DPR as compared to centralized connection routing.

To address these two goals we ran two series of experiments. The first series used a small network in determining the performance overhead of Stateful DPR and Stateless/LAN DPR when compared to a centralized connection router, and to baseline cases involving no connection routing. For this set of experiments we used the SPECweb96 [6] benchmarking tool because it places relatively smooth loading on the server over time.

The second series of experiments concentrated on exploring the scalability of Stateful DPR compared to centralized connection routing. Since our goal in this section was to explore how DPR would behave under realistic conditions, we used the Surge reference generator [4] to provide the server workload. Surge is a tool developed as part of the COMMONWEALTH project that attempts to accurately mimic a fixed population of users accessing a Web server. It adheres to six empirically measured statistical properties of typical client requests, including request size distribution and inter-arrival time distribution. As a result, it places a much burstier load on the server than does SPECweb96. In addition, while SPECweb96 uses an open system model (requested workload is specified in GETs/sec), Surge adopts a closed system model (workload is generated by a fixed population of users, which alternate between making requests and lying idle). As a result, Surge's workload intensity is measured in units of User Equivalents (UEs).

In both series of experiments we restricted our configurations to a single LAN in order to provide repeatable results. Although the LAN was not completely isolated during our measurements, only a negligible amount of unrelated traffic (mostly ARP requests) was present.

4.1. Performance Overhead of DPR

As described above, SPECweb96's principal independent parameter is the requested throughput, measured in HTTP GETs per second. The measured results of each experiment are the achieved throughput (which may be lower than what was requested) and the average time to complete an HTTP GET (measured in msec/GET). For each experiment, we ran SPECweb96 for the recommended 5 minute warmup, after which measurements were taken for 10 minutes. System hosts (both clients and servers) consisted of Hewlett-Packard Vectra PCs, each having a 200MHz Pentium Pro processor, 32 MB of memory, and a SCSI hard drive. Servers ran Linux 2.0.30 on Linux ext2 filesystems, while clients ran Windows NT 4.0. We used the NT Performance Monitor to ensure that

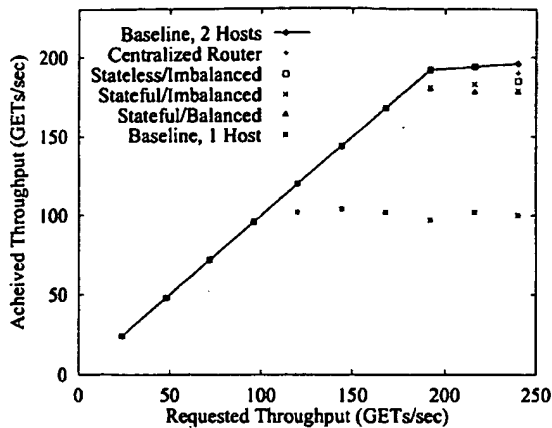


Figure 2. Throughput of DPR Variants

our clients had capacity to spare when our servers became saturated. The LAN was a 100 Mbit/sec Hewlett-Packard AnyLAN switch; this star network is frame-compatible with Ethernet, but it also uses a round-robin schedule together with a client sensing mechanism so that packet collisions do not occur. The Web servers used were Apache 1.2.4.

We describe the results of six experiments:

Baseline 1-Host. This experiment tests the performance of a single, unmodified server driven by a single client.

Baseline 2-Host. This experiment consists of two simultaneous copies of the Baseline 1-Host experiment. It uses two clients and two servers, and each client sends requests to only one server.

Centralized Router. This experiment consists of two clients sending requests to a centralized connection router, which then distributes the load evenly between two servers. The Centralized Router implementation is our Stateful DPR configured to redirect all connections to the other two servers (i.e. the routing function does not compete with local web service).

Stateless/Imbalanced. This experiment uses the Stateless/LAN variant of DPR, running on two hosts. Two clients generate requests, but they send *all* requests to one of the server hosts, which then redistributes half of them.

Stateful/Imbalanced. This experiment uses the Stateful variant running on two hosts. Again two clients generate requests, sending all requests to one host, which redistributes half of them.

Stateful/Balanced. This experiment again uses the Stateful variant, but now the two clients generate equal amounts of requests for each server host. Each host then redistributes half of its requests, sending them to the other server.

Baseline 1-Host and Baseline 2-Host define the range of possible performance for the systems under study, with Baseline 2-Host defining the best performance that might be expected

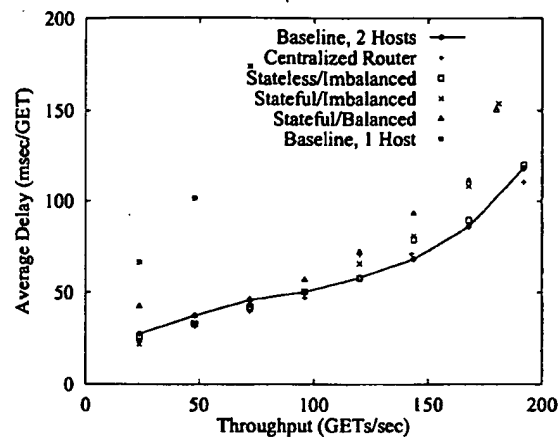


Figure 3. Request Delay of DPR Variants

from a 2-host server system. The Centralized Router results represent the performance of the most common alternative to DPR, and show the effect of removing the packet rewriting function from the server hosts. Note that each packet travels through two server nodes in the DPR and Centralized Router cases, and through only one server node in the Baseline cases.

The Stateless/Imbalanced and Stateful/Imbalanced experiments serve to show the worst possible performance of DPR, i.e., when the arriving request load is maximally imbalanced (all requests to one host). The Stateful/Balanced experiment allows comparison of the best and worst possible load arrival distributions for DPR.

Throughput:

In Figure 2 we show the achieved throughput of each experimental system as a function of the requested throughput. The Baseline 1-Host case saturates at about 100 GETs/sec, and the Baseline 2-Host case at the corresponding level of about 200 GETs/sec. In between the experiments fall into two groups: the Stateful experiments saturate at about 180 GETs/sec, while the Stateless/Imbalanced and Centralized Router saturate at about 195 GETs/sec. The fact that the Stateful/Balanced and Stateful/Imbalanced show nearly identical performance indicates that when requests arrive in a highly imbalanced way and all packet rewriting occurs on only one host, DPR is still able to achieve good throughput. This comparison indicates that the performance demand of packet rewriting is quite moderate, and so adding a packet rewriting function to a host already performing Web service does not represent a significant additional burden.

Comparing the Stateful and Stateless cases, we see that the Stateless case performs indistinguishably from the Centralized Router case, and they both are equivalent to the Baseline 2-Host case (in which no packet rewriting is taking place at all). The similarity of the Stateless to the Baseline 2-Host case shows that the performance cost of packet rewriting in

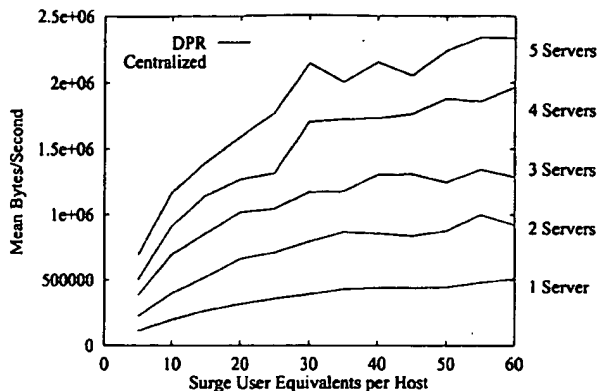


Figure 4. Throughput Comparison

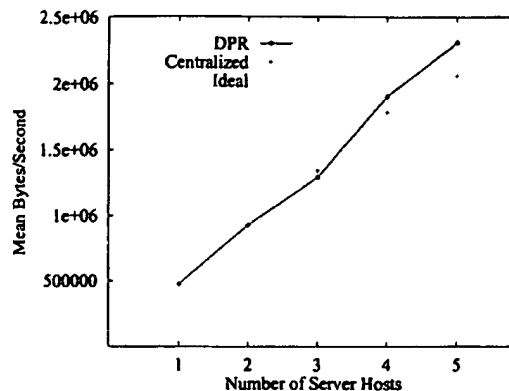


Figure 5. Scalability Comparison

the Stateless/LAN implementation is negligible.

An important implication of the similarity of the Stateless and Centralized Router cases is that the centralized connection routing architecture is not cost-effective. This is because an entire node has been allocated to the connection routing task (there are three nodes in the Centralized Router case and only two in the DPR cases). Thus the additional cost of adding a specialized connection router to a small system may not be justified. It is just as efficient, and cheaper, to use the server hosts already present to perform the connection routing function. This point will be reinforced by our results in Section 4.2 on the scalability of the DPR architecture compared to the centralized routing architecture.

Delay:

In addition to providing high throughput, it is important to verify that DPR does not add unacceptable delays to the system. In Figure 3 we show the average response time of an HTTP GET (in msec/GET) as a function of system throughput, for the same six experiments. In this figure we only plot those points for which achieved and requested throughput are nearly equal, so throughput does not reach quite the same maximum values as in Figure 2. Figure 3 shows that the experiments break into the same groupings as before. Again, the Stateful/Balanced and Stateful/Imbalanced cases show approximately similar performance. Furthermore the Stateless case shows approximately similar delays to the TCP Router and the Baseline 2-Host cases.

Since packets travel through an additional server node in the DPR and TCP Router cases as compared to the Baseline 2-Host case, there is a potential for greater delay in those cases. However, it appears that the additional delays induced by the additional hop are small compared to the average response time for an HTTP GET. The response time of an average HTTP GET under SPECweb96 is in the range of 25 to 150 milliseconds on a LAN. Were the system serving packets over the global Internet, response times would be even greater since the added round-trip times would be tens to hundreds of

milliseconds as well. The addition of additional packet processing due to Stateless/LAN DPR, which appears to be on the order of tens to hundreds of microseconds, is a negligible additional cost for a Web server application.

4.2. Scalability of DPR

The previous section showed that the overheads of DPR were no greater than that of a centralized connection router, and that even when connection routing load was completely unbalanced, system performance did not suffer. These results suggest that DPR should show good scalability, but it is still necessary to evaluate DPR's scalability in practice. For comparison purposes we also evaluate the centralized connection routing case.

The scalability series of experiments took place on different equipment than the performance overhead experiments. All of the Web/DPR servers were Dell Dimension PCs with 64 MB of memory and IDE hard drives running Linux 2.0.30 and Apache 1.2.1. Four of the Web servers had 200 MHz Pentium Pro processors, and one had a 233 MHz Pentium II. The latter system appears in our results as the fourth Web server in both DPR and centralized connection router experiments. Our fastest system, a 266 MHz Pentium II, was used only as a connection router. Both clients and servers used Linux ext2 file systems. The LAN was a 12-port 3Com SuperStack II Switch 3000 10/100 running Ethernet in full duplex at 100Mb/s. The total bandwidth measured during the experiments show that network capacity was not a limiting factor; we also observed that our clients were able to saturate our servers before reaching their own capacity.

As described above, for these experiments we used the Surge load generator. In the DPR cases, we configured Surge so that equal amounts of traffic were directed at each server host. In an N host system, each DPR host serves $1/N$ of the requests locally and distributes $(N - 1)/N$ of the requests equally to the other hosts in the system. In adopting this routing policy, our results for DPR are quite conservative.

A better policy that is still quite practical would be for each server host to only redirect requests that arrive when the host is loaded above the system average; in that case, a fraction of requests much smaller than $(N - 1)/N$ would be redirected, and the overall performance of the DPR system would be better than that reported here.

In our experiments we compare N -host DPR systems to centralized routing configurations consisting of N server hosts *plus* a connection router. By doing so, we emphasize the scalability difference between the architectures. However these tests do not compare equivalent systems in terms of hardware costs; as described above, it is more cost-effective to organize a system of N hosts in a DPR architecture than to set aside one host solely for connection routing.

In order to scale the demand placed on the server systems as the number of server hosts grows, it is necessary to proportionally increase the number of User Equivalents used in Surge. For this reason we report results in terms of User Equivalents per server host.

The achieved throughput for a range of both DPR systems and centralized routing systems is shown in Figure 4. This figure shows that for small systems (2-3 hosts), DPR and centralized routing behave approximately equivalently. However for larger systems (4-5 hosts), the centralized approach seems to show lower maximum throughput than the DPR approach. This evidence that the centralized node is beginning to become a bottleneck is supported by the fact that the difference between the two systems becomes more pronounced as the demand grows.

To illustrate the onset of a bottleneck effect in the centralized routing case, we show the peak throughput achieved as a function of the size of the system in Figure 5. Peak throughput was obtained in each case by averaging the throughput over the range 50-60 UEs per host (which in each case was where system saturation was judged to have set in). The figure also shows the "ideal" throughput obtained by simply scaling up the throughput obtained by a single unmodified host.

This figure shows that DPR obtains near-perfect speedup for server systems up to five hosts in size. In contrast, the centralized routing architecture seems to show signs of inefficiency at larger sizes; on four hosts, maximum throughput under centralized routing has dropped to 94% of ideal, and on five hosts the centralized system is only 86% efficient.

5. Summary

In this paper we have proposed and experimentally evaluated a protocol for routing connections in a distributed server without employing any centralized resource. Instead of using a distinguished node to route connections to their destinations, as in previous systems, Distributed Packet Rewriting (DPR) involves *all* the hosts of the distributed system in connection routing. The benefits that DPR presents over cen-

tralized approaches are considerable: the amount of routing power in the system scales with the number of nodes, and the system is not completely disabled by the failure of any one node. DPR allows more cost-effective scaling of distributed servers, and as a result more directly supports the goals of the COMMONWEALTH project.

References

- [1] D. Anderson, T. Yang, V. Holmedahl, and O. Ibarra. SWEB: Towards a Scalable World Wide Server on Multicomputers. In *Proceedings of IPPS'96*, April 1996.
- [2] E. Anderson, D. Patterson, and E. Brewer. The MagicRouter: An application of fast packet interposing. <http://HTTP.CS.Berkeley.EDU/~eanders/projects/magicrouter/osdi96-mr-submission.ps>, May 1996.
- [3] F. Baker. IETF RFC1812: Requirements for IP Version 4 Routers. See <http://ds.internic.net/rfc/rfc1812.txt>.
- [4] P. Barford and M. Crovella. Generating representative workloads for network and server performance evaluation. In *Proceedings of ACM SIGMETRICS '98*, pages 151-160, Madison, WI, June 1998.
- [5] S. Clarke. Port Forwarding in Linux. See description at <http://www.ox.compsoc.org.uk/~steve/portforwarding.html>.
- [6] T. S. P. E. Corporation. Specweb96. <http://www.specbench.org/org/web96/>.
- [7] D. M. Dias, W. Kish, R. Mukherjee, and R. Tewari. A scalable and highly available web server. In *Proceedings of IEEE COMPCON'96*, pages 85-92, 1996.
- [8] S. Garfinkel. The Wizard of Netscape. *WebServer Magazine*, pages 58-64, July/August 1996.
- [9] IBM Corporation. The IBM Interactive Network Dispatcher. See <http://www.ics.raleigh.ibm.com/netdispatch>.
- [10] E. D. Katz, M. Butler, and R. McGrath. A scalable HTTP server: The NCSA prototype. In *Proceedings of the First International World-Wide Web Conference*, May 1994.
- [11] K. Law, B. Nandy, and A. Chapman. A Scalable and Distributed WWW Proxy System. Technical report, Nortel Limited Research Report, 1997.
- [12] Linux IP Masquerade Resource. See <http://ipmasq.home.ml.org>.
- [13] J. Mogul, R. Rashid, and M. Accetta. The Packet Filter: An Efficient Mechanism for User-level Network Code. In *Proceedings of SOSP'87: The 11th ACM Symposium on Operating Systems Principles*, 1987.
- [14] J. C. Mogul. Network behavior of a busy Web server and its clients. Research Report 95/5, DEC Western Research Laboratory, Oct. 1995.
- [15] C. Perkins. IETF RFC2003: IP Encapsulation within IP. See <http://ds.internic.net/rfc/rfc2003.txt>.
- [16] R. J. Schemers. lbnamed: A Load Balancing Name Server in Perl. In *Proceedings of LISA'95: The 9th Systems Administration Conference*, 1995.
- [17] C. Systems. Scaling the Internet Web Servers: A white Paper. <http://www.cisco.com/warp/public/751/lodix/scale.wp.htm>, 1997.

Memex: A browsing assistant for collaborative archiving and mining of surf trails

Soumen Chakrabarti

Sandeep Srivastava

Mallela Subramanyam

Mitul Tiwari

Indian Institute of Technology Bombay

soumen,sandy,manyam,mits@cse.iitb.ernet.in

Abstract

Keyword indices, topic directories, and link-based rankings are used to search and structure the rapidly growing Web today. Surprisingly little use is made of years of browsing experience of millions of people. Indeed, this information is routinely discarded by browsers. Even deliberate bookmarks are stored in a passive and isolated manner. All this goes against Vannevar Bush's dream of the *Memex*: an enhanced supplement to personal and community memory.

We propose to demonstrate the beginnings of a 'Memex' for the Web: a browsing assistant for individuals and groups with focused interests. Memex blurs the artificial distinction between browsing history and deliberate bookmarks. The resulting glut of data is analyzed in a number of ways at the individual and community levels. Memex constructs a topic directory customized to the community, mapping their interests naturally to nodes in this directory. This lets the user recall topic-based browsing contexts by asking questions like "What trails was I following when I was last surfing about *classical music*?" and "What are some popular pages in or near my community's recent trail graph related to *music*?"

1 Motivation

Three paradigms have emerged for exploring the Web: keyword search, directory browsing, and following links. Popular search engine and directory sites are visited tens of millions of times per day. We speculate that the total number of clicks per day is orders of magnitude larger. This third source of information, the browsing history of millions of Web users over several years, an information source that dwarfs the scale of the Web itself, is almost entirely discarded by browsers as 'history'. Deliberate 'bookmarks' are preserved, but passively, in browser-dependent formats; this separates them from the dominant world of HTML hypermedia, even if their owners were willing to share them (as they are, in our experience, with all but a small section of their browsing activity).

Permission to copy without fee all or part of this material is granted provided that the copies are not made or distributed for direct commercial advantage, the VLDB copyright notice and the title of the publication and its date appear, and notice is given that copying is by permission of the Very Large Data Base Endowment. To copy otherwise, or to republish, requires a fee and/or special permission from the Endowment.

Proceedings of the 26th VLDB Conference,
Cairo, Egypt, 2000.

In 1945, Vannevar Bush dreamt of *Memex*: an enhanced, intimate supplement to personal and community memory [2]. Assisted by a Memex for the Web, a surfer can ask:

- What was the URL I visited about six months back regarding compiler optimization at Rice University?
- What was the Web neighborhood I was surfing the last time I was looking for resources on classical music?
- Are there any popular sites, related to my (Web) experience on classical music, that have appeared in the last six months?
- How is my ISP bill divided into access for work, travel, news, hobby and entertainment?
- What are the major topics relevant to my workplace? Where and how do I fit into that map? How does my bookmark folder structure map on to my organization?
- In a hierarchy of organizations (by region, say) who are the people who share my interest in recreational cycling most closely and are not likely to be computer professionals?

Since Bush proposed Memex, the theme of a 'living' hypermedia into which we "weave ourselves" has been emphasized often, e.g., by Douglas Engelbart¹ and Ted Nelson², and of late by Tim Berners-Lee³ and Jim Gray⁴. Indeed, the current cost/volume ratio of storage makes it unnecessary to delete *anything* from one's Web surfing experience, provided we can make fruitful use of it.

We propose an architecture of a 'Memex' for the Web which can answer the above questions. Memex is a large project involving hypertext data mining, browser plug-in and applet design, servlets and associated distributed database architecture, and user interfaces. We have validated the design using a prototype implementation that we describe here. Memex is currently implemented on Netscape 4.5+. We are currently testing Memex with the help of local volunteers. The Memex service will be made

¹<http://jefferson.village.virginia.edu/elab/hf10035.html>

²<http://www.sfc.keio.ac.jp/~ted/>

³<http://www.w3.org/1999/04/13-tbl.html>

⁴http://research.microsoft.com/~gray/papers/MS_TR_99_50_TuringTalk.pdf

publicly accessible⁵. Further details about Memex have been reported elsewhere [4].

2 Client architecture overview

Memex should run on popular browsers. It should be possible to distribute updates and new features effortlessly to users. Hence the Memex client has been designed as an applet. In view of secure firewalls, proxies, and ISPs' restrictions on browser setups, the client should communicate with the server over HTTP. The data transferred should be encrypted, if desired, to preserve privacy.

The user can log on to a Memex server at the level of a department, organization, interest group, ISP, nation (or the world). The architecture makes no assumptions about the logical community level at which Memex might be deployed. At any time, the user can choose not to archive surfing actions, archive for private use, or archive for use by the community (Figure 1). If permitted, Memex taps the browser to get the current location and passes this on to the server, which then processes it in many ways.

Apart from a standard full-text search over all pages visited, the Memex client has several function tabs to assist topic-based mining. The editable **folder** tab (Figure 1) provides topic management: this is the means by which users exemplify their interests. Existing bookmarks from Netscape or Explorer can be imported into Memex's editable tree-structured topic view; conversely Memex can export back to these browsers. Apart from implicit history logging, bookmarks can be added to folders while surfing. A user will typically assign a bookmark explicitly to a topic. These assignments are analyzed by the server, which then classifies all surfed pages automatically into these folders. The folder tab can also be used to reinforce or correct the classifier. Memex also uses unsupervised clustering to propose a topic hierarchy [6] over a set of links that the user may want to reorganize. Periodically, the server consolidates all users' public folders and browse history into a topic directory tailored to the needs of that specific community (see §4 and Figure 4).

Users surf on many topics with diverse priorities. Because browsers have only a transient context (one-dimensional history list), surfers frequently lose context when browsing about a topic after a time lapse. Studies have shown that visiting Web pages is best expressed using spatial metaphors: your context is "where you are" and "where you are able to go" next [9]. Memex's topic classifier also helps render the topic-focused **trail** tab (Figure 2). In the trail tab, the left panel shows the user's topic folders. Selecting a folder replays the hypertext graph of recent pages publicly surfed by the community which are

⁵<http://www.cse.iitb.ernet.in/~soumen/memex/>

most likely to belong to the selected topic, and thus recreates the user's browsing context.

3 Server architecture overview

On the server side, the system should be robust and scalable. It is important that the server recovers from network and programming errors quickly, even if it has to discard a few client events. The server consists of servlets that perform various archiving and mining functions as triggered by client action, or continually as demons. We prefer servlets to CGI scripts because the client-server interactions exchange complex objects and sometimes have state. We prefer HTTP tunneling also because direct JDBC connections may be refused by many firewalls.

Server state is managed by two storage mechanisms: a relational database (RDBMS) such as Oracle or DB2 for managing metadata about pages, links, users, and topics, and a lightweight Berkeley DB⁶ storage manager to support fine-grained term-level data analysis for clustering, classification, and text search. Storing term-level statistics in an RDBMS would have overwhelming space and time overheads.

An interesting aspect of the Memex architecture is the division of labor between the RDBMS and the lightweight storage manager. Planning the architecture was made non-trivial by the need for asynchronous action from diverse modules. There are some user interface-related events that must be guaranteed immediate processing. Typically these are generated by a user visiting a page, or deliberately updating the folder structure. With many users concurrently using Memex, the server cannot analyze all visited pages, or update mined results, in real time. Background demons continually fetch pages, index them, and analyze them w.r.t. topics and folders. The data accesses made by these demons have to be carefully coordinated. This would not be a problem with the RDBMS alone, but maintaining some form of coherence between the metadata in the RDBMS and several text-related indices in Berkeley DB required us to implement a loosely-consistent versioning system on top of the RDBMS, with a single producer (crawler) and several consumers (indexer and statistical analyzers). Figure 3 shows a block diagram of the system.

4 Mining algorithms overview

The stream of data from surfers has to be analyzed in various ways. Some parts of the processing, such as keyword indexing, are mundane. Other parts constitute new algorithms or novel implementations.

For clustering we started with a bottom-up hierarchical agglomerative approach [6]. For classification we started with a Bayesian classifier [3]. Although these simple text-based techniques work reasonably well for

⁶<http://www.sleepycat.com>

average Web pages, bookmarked URLs offer special challenges: people tend to bookmark many "front pages" with less text and more graphics compared to typical Web documents. Surfers may also place two URLs in the same folder for functional reasons, even if the corresponding documents are syntactically dissimilar.

We have implemented two new learning algorithms for Memex. For classification we use a new technique that combines features from text, hyperlink and folder placement to offer significantly boosted accuracy, increasing from a mere 40% accuracy for text-only learners to about 80% with our more elaborate model.

We generalize clustering to finding a new notion of themes among the bookmarks. In principle, each user need not design his/her own topic hierarchy, given there are 'standard' ones like Yahoo!⁷ and the Open Directory⁸. In practice, these 'universal' hierarchies are neither necessary nor sufficient for individual surfers and focused communities, they are too specialized in most topics, and not sufficiently specialized in the areas in which the community is deeply interested. We propose a new formulation for discovering a topic hierarchy specifically expressing and addressing the interests of the community, refining topics where needed and coarsening where possible. Details of the new classification and theme discovery algorithms are reported elsewhere [4] (also see Figure 4).

Once topic hierarchies for the user community are determined, automatic resource discovery is undertaken by demons to update users about recent and/or authoritative sources, organized by topic [5]. 'Normalizing' all members of the community to themes also lets us represent surfers' interests in a *canonical form*: roughly speaking, a user profile is a set of weights associated with each node of a theme hierarchy; this gives us a means of comparing profiles that is far superior to overlap in sets of URLs. We intend to use this for better collaborative recommendation [10].

5 Related work

Our work is closest in spirit to two well-known systems, PowerBookmarks⁹ and the Bookmark Organizer [8].

PowerBookmarks is a semi-structured database application for archiving and searching bookmark files via explicit CGI programs. PowerBookmarks uses Yahoo! for classifying the bookmarks of all users. In contrast, Memex preserves each user's view of their topic space, and reconciles these diverse views at the community level. Furthermore, PowerBookmarks does not use hyperlink information for classification or for synthesizing themes. The Bookmark Organizer is a client-side solution for personal organization, but does

not provide community-level themes or topical surfing contexts. Purple Yogi¹⁰ is a client-side software which logs pages visited and clusters them into folders. Then it tunes in on the Purple Yogi server to collect additional related material. No community-level mining is involved; Purple Yogi explicitly guarantees that user-specific data is stored locally on the user's desktop and never shipped out. Thus scope for valuable collaboration is lost and surfing history becomes inaccessible from other places from which the user might browse.

Other Internet start-ups have been quick to discover the annoyance of surfers maintaining multiple bookmark files and the opportunity of a central, networked bookmark server. We can list several sites which, using Javascript or a plugin, import existing Netscape or Explorer bookmarks and thereafter lets the surfer visit their Web site and maintain it using CGI and Javascript: Yahoo Companion¹¹, YaBoo¹², Baboo¹³, Bookmark Tracker¹⁴, and Backflip¹⁵ are some examples. Some services like Third Voice¹⁶ enables surfers to attach public or private annotations to any page they visit. These are essentially glorified FTP services with none of our extensive server-side analysis.

Several visualization tools have been designed recently that explore a limited radius neighborhood and draw clickable graphs. These are often used for site maintenance and elimination of dead links. Mapuccino and Fetuccino from IBM Haifa are well known examples [7, 1]. Our context viewer could benefit from better hypertext rendering techniques.

References

- [1] I. Ben-Shaul, M. Herscovici, M. Jacovi, Y. S. Maarek, D. Pelleg, M. Shtalheim, V. Soroka, and S. Ur. Adding support for dynamic and focused search with Fetuccino. In *8th World Wide Web Conference*. Toronto, May 1999.
- [2] V. Bush. As we may think. *The Atlantic Monthly*, July 1945. Online at <http://www.theatlantic.com/unbound/flashbks/computer/bushf.htm>.
- [3] S. Chakrabarti, B. Dom, R. Agrawal, and P. Raghavan. Scalable feature selection, classification and signature generation for organizing large text databases into hierarchical topic taxonomies. *VLDB Journal*, Aug. 1998. Invited paper, online at http://www.cs.berkeley.edu/~soumen/VLDB54_3.PDF.
- [4] S. Chakrabarti, S. Srivastava, M. Subramanyam, and M. Tiwari. Archiving and mining community web browsing experience using Memex. In *9th International World Wide Web Conference*, Amsterdam, May 2000.
- [5] S. Chakrabarti, M. van den Berg, and B. Dom. Focused crawling: a new approach to topic-specific web resource discovery. *Computer Networks*, 31:1623-1640, 1999. First appeared in the *8th International World Wide Web Conference*¹⁷.

¹⁰<http://www.purpleyogi.com>

¹¹<http://www.yahoo.com/r/cm>

¹²<http://www.yaboo.dk>

¹³<http://www.baboo.com>

¹⁴<http://www.bookmarktracker.com>

¹⁵<http://www.backflip.com>

¹⁶<http://www.thirdvoice.com>

¹⁷<http://www8.org>

⁷<http://www.yahoo.com>

⁸<http://dmoz.org>

⁹<http://www.ccrl.necelab.com/webdb/>

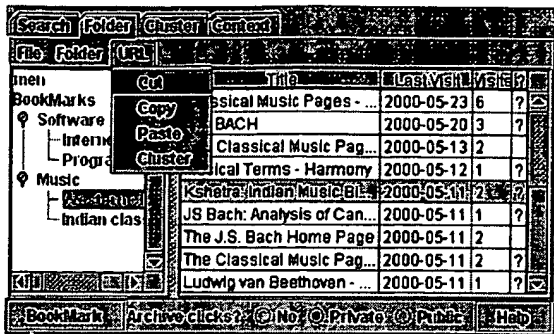


Figure 1: Each user has a personal folder/topic space, which is usually initialized by importing existing browser-specific bookmark folders. The classification demon then classifies all subsequent history elements, marking its guesses by '?'. The user can correct or reinforce the classifier using cut/paste, thus continually improving Memex's models for the user's topics of interest.

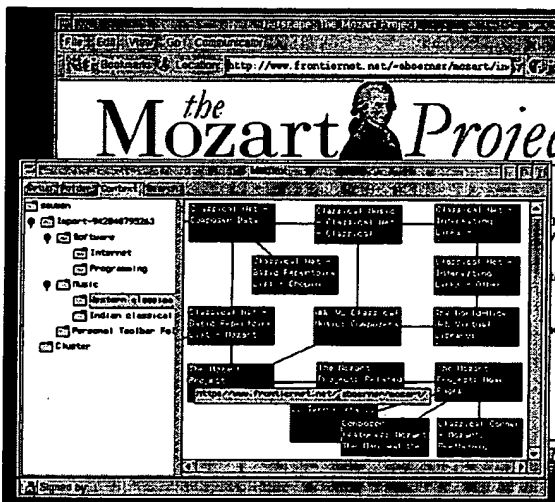


Figure 2: The trail tab shows a read-only view of the user's current folder structure. When the user selects a folder, Memex replays recently browsed pages which belong to the selected (or contained) topic(s), reminding the user of the latest topical context. In the screen-shot above, the chosen folder is /Music/Western Classical. The user can now resume browsing and the display is updated with additional resources related to the topic.

Toronto, May 1999. Available online at <http://www8.org/w8-papers/5a-search-query/crawling/index.html>.

- [6] D. R. Cutting, D. R. Karger, and J. O. Pedersen. Constant interaction-time scatter/gather browsing of very large document collections. In *Annual International Conference on Research and Development in Information Retrieval*, 1993.
- [7] M. Hersovici, M. Jacovi, Y. S. Maarek, D. Pelleg, M. Shtalheim, and S. Ur. The Shark-Search algorithm-an application: Tailored web site mapping. In *7th World-Wide Web Conference*, Brisbane, Australia, Apr. 1998. Online at <http://www7.scu.edu.au/programme/fullpapers/1849/com1849.htm>.

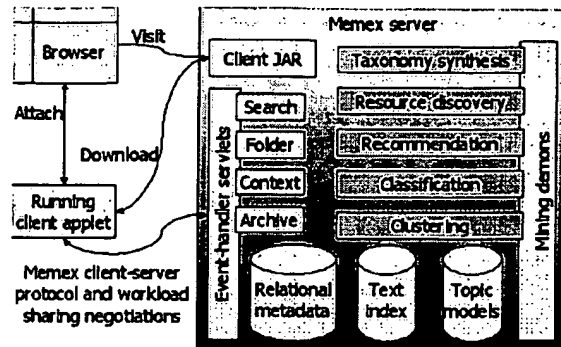


Figure 3: Block diagram of the Memex system, showing the client-server interface, the UI event handlers, the mining demons, and the loosely synchronized data repositories.

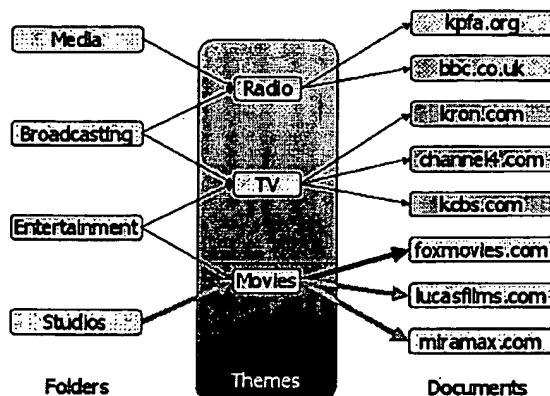


Figure 4: Memex computes, from the document-folder associations of multiple users, a topic taxonomy specifically tailored for the interests of that user population. The taxonomy consists of *themes* which capture common factors in people's interests when they can, while maintaining individuality when they must. Once computed, they can be used to guide resource discovery and collaborative recommendation.

- [8] Y. S. Maarek and I. Z. Ben Shaul. Automatically organizing bookmarks per content. In *Fifth International World-Wide Web Conference*, Paris, May 1996.
- [9] P. P. Maglio and T. Matlock. Metaphors we surf the Web by. In *Workshop on Personalized and Social Navigation in Information Space*, Stockholm, Sweden, 1998.
- [10] L. H. Ungar and D. P. Foster. Clustering methods for collaborative filtering. In *AAAI Workshop on Recommendation Systems*, 1998. Online at <http://www.cis.upenn.edu/~ungar/papers/clust.ps>.

APNNed goes Internet

P. Kemper and C. Tepper *

Dept. of Computer Science IV, University of Dortmund
D-44221 Dortmund, Germany
e-mail: {kemper, tepper}@ls4.cs.uni-dortmund.de

Abstract

APNNed is an editor for hierarchical, colored, and stochastic Petri nets. It is an integral part of the APNN toolbox and serves as a graphical user interface for other members of this toolset. Since it is implemented in Java, it is rather platform independent. In this paper we describe recent enhancements to support modeling and analysis of Petri nets with APNNed via internet. We give a brief sketch of technical issues like HTTP-tunneling that need to be solved to have an APNNed applet running at the client side and an APNNed servlet at the server side, that provides load, save, and analysis functionality for nets.

1 Introduction

A major attraction of web applications is its ease of use; a user need not install the specific application but can rely on a general purpose web browser possibly enriched by a set of plug-ins. Furthermore, web applications can be made accessible to an arbitrary or to a restricted set of users with limited effort. A Petri net tool can profit from these advantages in many ways, for example, for teaching modeling with Petri nets in classes, students can work with Petri net tools at their home PC, or for large research projects, in which different groups of researchers interact and need to use and provide tools or applications for each other. However, few Petri net tools make use of this opportunity yet.

In this paper, we briefly describe a recent attempt to enhance the java-implemented Petri net editor APNNed [7] of the APNN toolbox with functionality to perform as a web application. We consider a scenario, in which APNNed applet is run by a web browser. The applet allows to load a set of available models from an HTTP server or to create new Petri net models. Any model can be edited and modified as in the stand-alone APNNed. Fig. 1 shows a screenshot of APNNed as an applet of an HTML page viewed by a browser. Note that, once started, applets are not restricted to the canvas space taken by the web browser but can expand over the whole screen. The applet provides some functionality like the token game by itself and more sophisticated analysis by help of a servlet that resides at the HTTP server and that uses the APNN toolbox. The servlet provides a load and save operation for persistent storage of nets. Obviously, the possibility to save models at the server's site makes sense only in a restricted group of users. In the following, we will not focus on security but on some technical issues to set up the necessary communication among applet and servlet to transfer information on nets, requests for analysis and analysis results. During its implementation, we had to recognize that

*This research is supported by DFG, collaborative research center 559 'Modeling of Large Logistic Networks'.

this case is not standard and among the many Java books that discuss servlets we found only in [10] a substantial treatment of HTTP-tunneling. Since many Petri net tool developers may consider to turn their tools into web applications, we believe that our experiences are useful and such technical issues are of broader interest.

Before we go into these details, let us briefly recall the functionality and architecture of the APNN toolbox. The APNN Toolbox [2] is a collection of analysis tools, all of which are combined by APNNed [7], which serves as a graphical user interface to edit Petri net models and also controls application of analysis tools for a given net, such that results of analysis are subsequently visualized. All tools interact via a textual interface called Abstract Petri net notation (APNN) [3]. The APNN toolbox contains programs for the functional and quantitative analysis. Functional analysis includes the calculation of invariants, based on the algorithm of Martinez and Silva [13], a randomized token game to detect deadlocks, a analysis techniques based on the reachability graph like deciding liveness and modelchecking. Model checkers exists for a computational tree logic (CTL) [12] and a linear temporal logic (LTL). Quantitative analysis is supported by a discrete event simulator and numerical analysis methods based on an associated continuous time Markov chain (CTMC). The APNN toolbox has a clear focus on state based analysis, in which extremely large but finite state transitions systems are represented by hierarchical or modular Kronecker representations [4, 5, 6, 11]. State-based analysis tools interact via a second, a state level interface that describes the reachability set as a multi-way decision diagram and the reachability graph as a set of matrices that are composed by Kronecker operations.

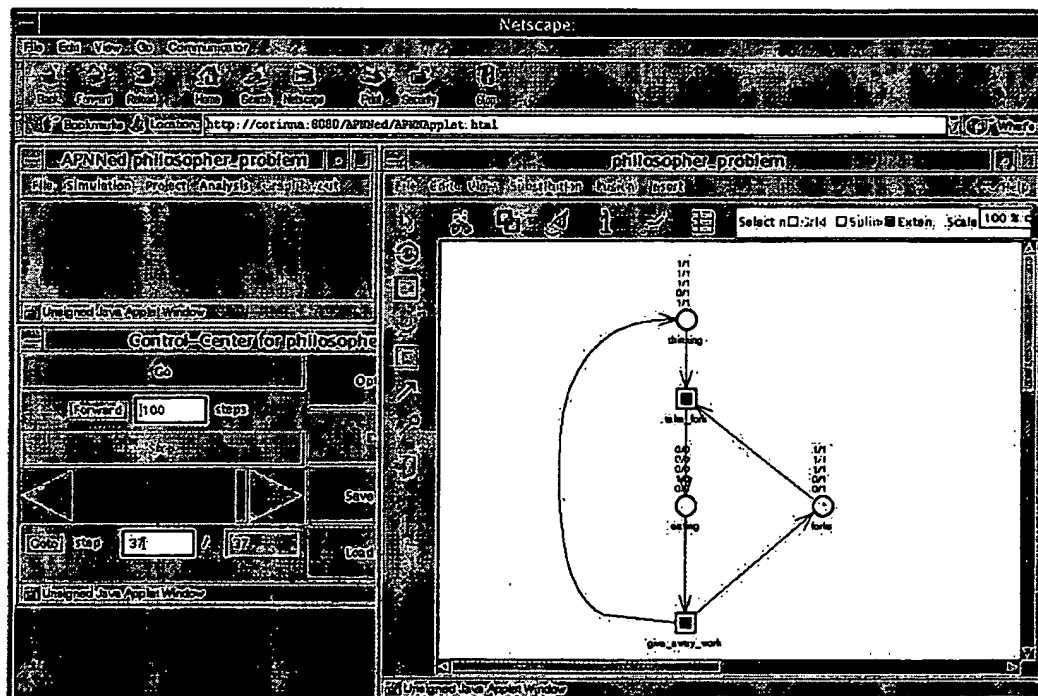


Figure 1: Screenshot of the internet version of APNNed

The paper is structured as follows. Section 2 describes design considerations and identifies four basic operations to achieve an internet version of APNNed. Section 3 describes how to set up an applet-servlet communication (HTTP-tunneling) for an applet, while Section 4 gives the counterpart for the servlet. We summarize in Section 5.

2 Design issues

Web applications are often implemented by Common Gateway Interface (CGI) programs such that a HTTP request of a client results in the instantiation of a process at the server's site to generate a response which often is a web page in HTML. Starting a new process for each HTTP-request causes a significant overhead if the response computation itself only takes a minor effort. A servlet is a similar concept in Java but it runs on threads rather than processes with obvious advantages for performance and the ability to exchange data among threads, such that a servlet can store data over a sequence of requests from a client. Java servlet technology [10] provides web developers with a simple mechanism for extending the functionality of a web server. Servlets are server- and platform-independent and have access to the entire Java APIs [8], [9]. A servlet profits from Java's general properties like portability, performance, re-usability and crash protection (to the extend an existing virtual machine and language implementation can keep pace with expectations).

APNNed is implemented in Java, so that it is a natural choice to use servlets. As mentioned above, our scenario focuses on four simple operations, namely to load, save and analyze nets at the server's site and an additional operation to give information on which nets are available at the server. We do not consider additional topics imposed by storing nets by the server, which nevertheless need to be considered in practice, these include security, restricted access, privacy and sharing of nets, but also traversal of directory structures, disc usage and cpu usage by external clients for analysis etc.

We focus on the applet-servlet communication. The four operations require transfer of an operation identifier plus operation specific parameters as given below:

Operation	Parameter1	Parameter2	Parameter3
load	operation=load	netname=Example	
save	operation=save	net=<APNN description>	netname=Example
analyze	operation = analyze	netname=Example	method=name+parameters
directory	operation=dir		

The information for each operation need to be encoded at the sender and decoded by the receiver. Parameters are rather straightforward, i.e. netname gives the filename of the net, net gives a textual description of the net in APNN format, method gives an identifier for the selected analysis method plus some additional method-specific parameters.

Since applet and servlet are both Java applications, one can transfer serialized Java objects, such that a receiver can pick the complete information for an object just by a single call to *readObject* and one need not programme the parsing and assignment procedure to instantiate an appropriate object. Alternatively, applet and servlet can communicate via buffered input stream to transfer binary or ASCII data. In both cases, an Java *URLConnection* is needed to establish an input stream. The internet version of APNNed uses an object input stream to send and receive objects of class *String*. Clearly, we would use an object stream to exchange net objects but since some internal APNNed classes to represent a net are not serializable yet, we

need to transfer an APNN description as a string object and the receiver needs to parse this string. The encoding of an operation into a string uses an `URLencoder` object, e.g., for the load operation it is: `String data = "operation=" + URLencoder.encode("load")`
`+ "netname=" + URLencoder.encode(<name of Petri net>);`
 Encoding for other operations is done accordingly.
 In the following, we present the necessary steps to call the operations above by an applet and to transfer corresponding answer by an servlet.

3 Starting communication by a client

The applet sends first data to the servlet before the servlet sends back any kind of response. To start the communication, an applet mainly needs to open a connection with an `URLConnection` object and to generate a request with an appropriate output stream. The steps given below give the details and clearly identify the Java classes that are involved, for more information see [10]. The steps need to be integrated in a try/catch block to account for exceptions.

- 1.) Create a `URL` object which references to the home of the applet
`URL currentPage = getCodeBase();`
`String protocol = currentPage.getProtocol();`
`String host = currentPage.getHost();`
`int port = currentPage.getPort();`
`String urlSuffix = "/servlet/SimpleServlet";`
`URL dataURL = new URL(protocol, host, port, urlSuffix);`
- 2.) Create a `URLConnection` object
`URLConnection connection = dataURL.openConnection();`
- 3.) Prohibit the web browser to cache the URL data
`connection.setUseCaches(false);`
- 4.) Disclaim the system to allow read and write of data
`connection.setDoOutput(true);`
- 5.) Create a `ByteArrayOutputStream` to buffer the data that should be sent to the servlet
`ByteArrayOutputStream byteStream = new ByteArrayOutputStream(512);`
- 6.) Connect the `ByteArrayOutputStream` with the output stream `PrintWriter`
`PrintWriter out = new PrintWriter(byteStream, true);`
- 7.) Put the data for the server in the buffer
`String data = "param1=" + URLencoder.encode(value1)`
`+ "¶m2=" + URLencoder.encode(value2);`
`out.print(data);`
`out.flush();`
- 8.) Set the *Content-Length*-Header
`connection.setRequestProperty("Content-Length",`
`String.valueOf(byteStream.size()));`
- 9.) Set the *Content-Type*-Header
`connection.setRequestProperty("Content-Type",`
`"application/x-www-form-urlencoded");`
- 10.) Send the data to the servlet
`byteStream.writeTo(connection.getOutputStream());`

The servlet sends answers through a response object that connects itself to the connection object that was instantiated for the request. Hence the applet needs to set up an input stream on this connections in order to receive the answer of the servlet. The following steps are necessary for this task.

- 11.) Create an `ObjectInputStream`
`ObjectInputStream in = new ObjectInputStream(connection.getInputStream());`
- 12.) Read data (*String*) with the method `readObject`
`String in = (String)in.readObject();`
- 13.) Close input stream
`in.close();`

In order to implement the applet, we made only minor extensions to APNNed to implement the communication procedures above and to avoid any kind of access to the local file system. It remains to describe the servlet side of the communication.

4 Responding to client request by a servlet

The servlet needs to take care of incoming requests by generating a new thread for each request. In this thread, the servlet must decode the message and interpret its content to trigger appropriate actions for a response. A servlet extends the class `HttpServlet`, its most relevant methods are `init`, `service`, `doGet`, and `doPost`. The last three functions have two parameters, a request object of class `HttpServletRequest` and a response object of class `HttpServletResponse`. The request object allows to access incoming HTTP-headers or data by opening an input stream. The response object allows to send HTTP-status codes and data by opening an output stream. Due to the similarities between receiving and sending, we describe only how to send data to an applet, the servlet opens an output stream using the response object. The servlet must only realize 4 steps to send serialized data to an applet.

- 1.) Set the content type for sending a seralized data object
`String contentType = "application/x-java-serialized-object";`
`response.setContentType(contentType);`
- 2.) Create an `ObjectOutputStream`
`ObjectOutputStream out = new ObjectOutputStream(response.getOutputStream());`
- 3.) Write data with `writeObject` into the output stream
`out.writeObject(new String(<APNN description>));`
- 4.) The stream must be vacated. This ensures that the client get the complete data
`out.flush();`

Clearly, in addition to communication, the servlet implementation causes more work and code to establish the required functionality. Especially the control of the operations needs more considerations, for instance, the selection of analysis methods, its start, termination detection and return of results since the different analysis tools of the APNN toolbox are independent executables that are implemented in C and perform as independent processes. A sophisticated servlet implementation needs to take care of security, privacy, sharing among different users and applets; it need to take care of the usage of disc space, memory and cpu to avoid misuse and to possibly report on failures.

5 Conclusion

We briefly present the internet version of APNNed which supports loading, storing, and analysis of Petri nets via an HTTP server, so that APNNed can run as an applet in a general purpose web browser. Instead of describing the rather unchanged usage and appearance of APNNed, we describe how the necessary applet-servlet communication (HTTP-tunneling) can be realized in Java. Clearly, the current implementation covers a lot more details, for instance, transferring hierarchical nets implies to send a set of nets and information on its connectivity instead of a single net description. The current status of APNNed and the APNN toolbox is available at [1]. Ongoing work considers asynchronous interaction with time consuming analysis tools and further extensions to allow for a secure usage in an open environment.

References

- [1] <http://ls4-www.cs.uni-dortmund.de/qm/werkzeuge.html>.
- [2] F. Bause, P. Buchholz, and P. Kemper. A toolbox for functional and quantitative analysis of DEFS. *Quantitative Evaluation of Comp. and Comm. Sys.*, pages 356–359, Springer LNCS 1469, 1998.
- [3] F. Bause, P. Kemper, and P. Kritzing. Abstract Petri net notation. *Petri Net Newsletter*, 49:9–27, 1995.
- [4] P. Buchholz. Hierarchical structuring of superposed GSPNs. In *Proc. 7th Int. Workshop Petri Nets and Performance Models (PNPM'97), St-Malo (France), June 1997*, pages 81–90. IEEE CS Press, 1997.
- [5] P. Buchholz and P. Kemper. Efficient computation and representation of large reachability sets for composed automata. Forschungsbericht 705, Fachbereich Informatik, Universität Dortmund (Germany), 1999.
- [6] P. Buchholz and P. Kemper. Modular state level analysis of distributed systems - techniques and tool support. In W.R. Cleaveland, editor, *Proc. 5th Int. Conf. on Tools and Algorithms for the Construction and Analysis of Systems (TACAS '99)*. Springer, LNCS 1579, 1999.
- [7] P. Buchholz, P. Kemper, and APNNed group. APNNed - a net editor and debugger within the APNN Toolbox. In J. Desel, P. Kemper, E. Kindler, and A. Oberweis, editors, *5. Workshop Algorithmen und Werkzeuge für Petri Netze*, Universität Dortmund, Fachbereich Informatik, Forschungsbericht Nr. 694:19–24, 1998.
- [8] David Flanagan. *Java in a nutshell: A Desktop Quick Reference*. O'Reilly, 3 edition, Nov 1999.
- [9] David Flanagan, Jim Farley, William Crawford, and Kristopher Magnusson. *Java Enterprise in a nutshell*. O'Reilly, May 1999.
- [10] Marty Hall. *Core Servlets and Java Server Pages*. Prentice Hall, June 2000.
- [11] P. Kemper. Numerical analysis of superposed GSPNs. *IEEE Trans. on Software Engineering*, 22(9), Sep 1996.
- [12] P. Kemper and R. Lübeck. Model checking based on kronecker algebra. Forschungsbericht 669, Fachbereich Informatik, Universität Dortmund (Germany), 1998.
- [13] J. Martinez and M. Silva. A simple and fast algorithm to obtain all invariants of a generalized Petri net. In C. Girault and W. Reisig, editors, *Application and Theory of Petri Nets*, Informatik Fachberichte 52, 1982.

Lightweight, Dynamic and Programmable Virtual Private Networks

Rebecca Isaacs
Computer Laboratory
University of Cambridge
Cambridge, UK
Rebecca.Isaacs@cl.cam.ac.uk

Best Available Copy

Abstract—A Virtual Private Network (VPN) that exists over a public network infrastructure like the internet is both cheaper and more flexible than a network comprising dedicated semi-permanent links such as leased-lines. In contrast to leased-line private networks, the topology of such a VPN can be altered on-the-fly, and its lightweight nature means that creation and modification can take place over very short timescales.

In a programmable networking environment, such VPNs can be enhanced with fine-grained customer control right down to the level of the physical network resources, allowing a VPN to be employed for almost any conceivable network service. This paper examines some of the issues present in the provision of programmable VPNs. In particular, automated VPN “design” is considered, that is, how a VPN description can be translated to a set of real physical resources that meets customer requirements while also satisfying the goals of the VPN Service Provider (VSP). This problem—the distribution of resource allocations across network nodes in an optimal manner—has relevance for other approaches to VPN provision such as differentiated services in the internet [1].

The work described in this paper was carried out using a programmable networks infrastructure based on the switchlets mechanism [2]. It shows that automated VPN creation resulting in a guaranteed resource allocation is a feasible procedure that works well for both the VSP and for the customer that has requested a VPN. The problems inherent in dynamic VPN reconfiguration are also briefly explored together with the methods by which these might be addressed.

I. INTRODUCTION

Until recently, the management and control functionality of networks has been tightly coupled to the network hardware, leading to a “closed” and restrictive environment. The provision of new network services, signalling protocols and other software has been exclusively the domain of network equipment manufacturers, often, but not always, under the auspices of international standards bodies. The drawbacks of this approach are widely recognised:

- Although the use of standards ensures interoperability, the process of defining them is a slow and often highly political exercise. It is not unusual that by the time a standard is produced, technological developments have made it largely obsolete.
- Due to their monolithic nature, standards can be heavyweight and unwieldy, further stifling the impetus to develop and quickly deploy new network services.
- Mechanisms for introducing innovative network services are invariably very limited. If the demand has not already been anticipated, it is unlikely the network will be able to cater for the service in the optimal manner, if at all.
- Close integration of the network control software and the internal network elements makes upgrades and bug fixes difficult and costly.

Many illustrations of these shortcomings can be cited with respect to commonly deployed networks. Examples include the awkward integration of intelligent network services into the

telephone network, and the requirement for some ten thousand lines of code on every switch for UNI/NNI signalling in ATM networks.

A. Programmable Networking

Programmable networking has been advocated as a means of addressing these issues. The main idea of this approach is that the network can be programmed, in other words its physical resources accessed and manipulated, through an open programmable interface. Such an interface allows control and management functionality to be devolved from the internal network hardware to external processors, hence separating the role of equipment vendors from that of software and service providers.

Potential advantages of programmable networking include the opening up of the network to third parties, the easy introduction of sophisticated and hitherto unanticipated network services, and significant speedups in the deployment of such services. However, in practice these benefits are hard to deliver and pose some challenging questions such as how programmable network interfaces should be defined, how much abstraction is called for, as well as performance, robustness and security issues. Many of these issues are under active consideration in the research community [3].

B. Virtual Private Networks

Virtual Private Networks (VPNs) that make use of existing (possibly public) network infrastructure are a way of procuring the equivalent of a private leased-line network that is relatively cheap and flexible. Permanently dedicated resources are not required, and the topology and size of the VPN can be altered as required. These VPNs are most commonly deployed on the internet, often to provide connectivity between corporate LANs, or to enable individual mobile users to access a private LAN. The establishment of internet VPNs does not require large overheads: tunnelling is used to provide routing and addressing, and security measures such as encryption of the VPN's traffic and authentication protocols prevent data tampering and unauthorised access.

Issues of network performance and guaranteed quality of service, as well as mechanisms for pricing and charging where appropriate (especially when multiple ISPs participate at the boundary of the VPN), have not as yet been fully addressed, although their importance is recognised. Current VPN product offerings either operate solely on IP networks, or over a small number of other protocols.

C. Programmable VPNs

The provision of VPNs in a programmable network environment takes the VPN concept a step further. It contains the means for a VPN to control not just its routing and addressing mechanisms, but any aspect of the underlying network resources that can be "programmed". Thus a VPN is not constrained to use a particular networking protocol.

Depending on the programmable interface used, the exposure of the network internals can guarantee VPN performance, and provide a low-level monitoring and charging mechanism for the VPN Service Provider (VSP). This scenario is possible when there is a means to isolate one VPN from another, so that each VPN can have exclusive control of and access to "its" resources. The switchlets concept [2] does exactly this by partitioning the physical resources and policing those partitions. An open control interface allows a VPN to access its partition without interference from any other VPN.

In general, resource partitioning combined with open network control has a number of benefits. Including:

- the removal of the need for a single instantiation of a prescribed control system (i.e. signalling mechanism or networking protocol)—each partition of the network can be controlled by a different system;
- the support for true multi-service networks where each type of service can operate in its appropriate environment, unaffected by other users of the network;
- the potential for a network operator to offer a *VPN Service*, in which VPNs comprising some subset of the available physical resource (in terms of both topology and capacity), can be acquired on demand and their allocated physical resources manipulated at will.

A VPN deployed in this environment can be extremely lightweight. Its existence could be as brief as a matter of minutes, and the network control software as minimal as required. An example is a VPN created purely for the duration of a video conference, running *service specific* control software tuned to support the requirements of video conferencing [4]. During the lifetime of such a VPN, its resource requirements might change, for example as a result of participants joining or leaving the conference, and this can be reflected in corresponding changes in its underlying physical topology and resource allocation.

This paper examines some of the issues involved in the provision of programmable VPNs, from the point of view of both the VSP and the VPN customer. These issues include:

- The tradeoff between expressiveness and simplicity for a VPN specification language that must cater for descriptions that range from the minimal to the comprehensive.
- The conflict between VSP goals of maximising resource usage, and customer demands for the best VPN to meet their needs.
- The theoretical intractability of finding an optimal VPN topology given these conflicting requirements.

The feasibility of automated VPN creation has been investigated by observing how VPN topology search performs with a

naive implementation, and to what extent a simple heuristic improves performance. The scenario of a VSP using the switchlets infrastructure provides the context and the motivation for this work and is described in Section II. Section III proposes an approach to automated VPN design, encompassing a specification mechanism that reconciles the goals of the two parties to produce a cost function that is subsequently used to compute a VPN topology. Section IV presents the implementation experience, and further work concerning the problems inherent in dynamic reconfiguration of VPNs is discussed in Section V.

II. CONTEXT

A. Infrastructure

The infrastructure over which dynamic VPNs are provided is based on open signalling concepts. Control of the physical network is, as far as possible, devolved from the internal elements to general-purpose workstations. The functionality of the switch or router is encapsulated in an open control interface—typical operations available through such an interface include connection management, routing, alarm notification and the gathering of statistics. Examples of open switch control interfaces include GSMP [5] and VSI [6]. We use an interface developed in the Computer Laboratory called Ariel [7], [8].

Partitioning of the physical network is achieved by subdividing the resources of individual switches into one or more logically separate *switchlets*. Each of these is presented to its owning control system through an open switch control interface, which gives the illusion that the control system is managing an entire switch. This is the crucial feature that allows multiple control systems to co-exist on a single physical network, whilst also giving them fine-grained control of the resources they have been allocated.

A process called the *Divider* manages the creation, deletion and modification of switchlets. The Divider runs on a general-purpose workstation, ideally in a resource-controlled environment such as that provided by the Nemesis operating system [9]. Invocations on switchlet interfaces are policed on the control path by a Divider (one per switch) to ensure there is no interference between control systems. Fig. 1 shows multiple control systems sharing the resources of the physical network. Each control system makes invocations on the Ariel interface exported by the Divider for the corresponding resource allocation on that switch.

A full-blown virtual network can be constructed by acquiring switchlets in one or more network nodes. The *Network Builder* is responsible for the creation and deletion of virtual networks, and for allocating the resources required by those virtual networks. The topology of a virtual network need not map directly onto the underlying physical network. A single-link virtual network can span multiple physical links simply by reserving resources on all intervening nodes without exporting Ariel interfaces—such resource reservations are termed *tunnel switchlets*.

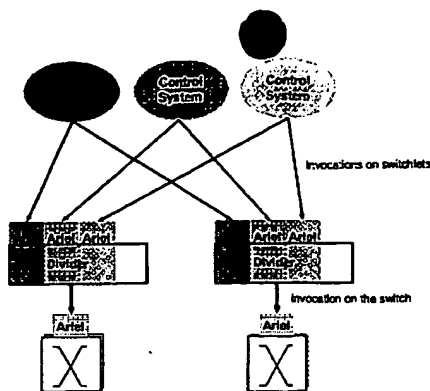


Fig. 1. Control systems operating simultaneously.

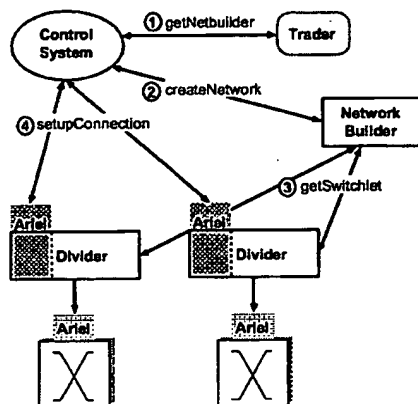


Fig. 2. Acquisition of a virtual network.

Communication between these architectural components takes place using a DPE, which runs over its own virtual network. A bootstrap virtual network is employed at start of day.

Fig. 2 illustrates the steps involved in the acquisition of a virtual network by a control system: a control system locates the Network Builder using a trader, and then asks it for a new network comprising some specified resources. The Network Builder in turn locates the Divider at each of the constituent nodes, and requests a new switchlet at each. After creating a switchlet, the Divider returns a new Ariel interface for that switchlet to the Network Builder. The switchlet interfaces are then passed back to the control system, which can then make invocations on switchlets directly and hence control its partition of the physical network resources.

The operation of control systems across possibly non-cooperating intervening networks is also addressed within the infrastructure [10]. Virtual networks can be set up that span multiple domains, with the result that a control system need not necessarily be aware of the underlying administrative boundaries. The infrastructure will take care of the tunnelling of traffic across other types of network, albeit with a potential loss of

service guarantees.

The provision for multiple control systems to operate simultaneously means that no single system need be prescribed for all users (which is not to say that multiple instances of the same control system can not co-exist). Although for many users standard, general-purpose control systems will suffice, others are able to run service-specific control systems tailored to their individual needs, if they wish to do so.

An implementation of this control framework is currently operational on an ATM network consisting of 5 ATM switches, 5 host workstations and 7 audio/video codecs. Ongoing developments include novel and innovative control systems, federated virtual networks over wide-area links, investigation into pricing and charging mechanisms, dynamic resource reallocation and adaptive control systems, and control system interoperability. For a general overview and more information see [7] and [8].

B. Related Work

Virtual networks within a programmable networking environment have been explored from two different angles. In *active networks* [11] programs can be inserted into the routers or switches and then executed on the messages passing through those nodes. This form of active network is the most extreme form of network programmability, but because the control and data paths are not distinguished it is difficult to provide hard guarantees for differentiation of services.

In contrast, the alternative approaches, which include the switchlets approach, partition network resources among virtual networks and provide some mechanism for independent operation of the virtual networks. Schemes such as Genesis [12] and VNRM [13] deploy a specific control system on a virtual network by instantiating objects that implement the desired control interface and protocol.

Using switchlets, no such built-in functionality is provided, but rather a handle onto a subset of the real, physical resources, which enables low-level and fine-grained manipulation of those resources. In consequence, a virtual network instantiated by means of a switchlet will be more lightweight and hence amenable to the automated VPN design that is the subject of this paper.

As an aside, direct manipulation of physical resources does not preclude the use of off-the-shelf control systems in a switchlets virtual network (indeed we run a cut-down version of IP on one of our virtual networks). It also does not mandate a particular resource abstraction for control, hence avoids needlessly restricting operations on the resource, or compromising efficiency, flexibility and performance.

Genesis addresses the automated deployment of virtual networks and their control systems through a process called spawning. This uses a network blueprint in the form of an executable profiling script which lists requirements for addressing, routing, management, topology, resources and so on. The profiling script is produced manually by a network architect, and the conflicting goals of VSP and virtual network customer that

are the subject of this paper are not addressed. Although Genesis has not as yet been implemented, the implementation plan described in [12] intends to tackle some of these issues.

Ongoing work concerning resource management in VPNs includes the "hose model" [14], in which a performance abstraction for IP-based VPNs is proposed that characterises the desired performance characteristics of a VPN by an aggregated capacity figure. Extensive evaluation using trace driven simulations shows that considerable benefit is gained by statistically multiplexing traffic across the VPN as a whole—a technique that can also be employed in the context of this work where appropriate because of the hard partitioning of resources between VPNs.

Proposals for differentiated services in the internet [1] effectively partition resources in a public network in a similar way. However, mechanisms for dynamically shifting resources from one aggregate traffic classification to another have not as yet been defined, although the desirability of such behaviour is recognised.

C. A VPN Service Provider

The nature of the issues considered in this paper is motivated by the envisaged requirements of a VPN Service Provider (VSP) making use of the infrastructure to sell dynamic and flexible VPNs to a range of customers. A VSP will benefit from the potential for efficient use of network resources, multiplexing gains inside the network, and flexible and fast response times to customer demands. The VSP may also offer extra value over the standard VPN service, for example with configurable reliability for VPNs, advance VPN reservations or the facility for dynamically loadable customer code inside a switchlet (i.e. extremely close to the physical network itself). A flexible and easy-to-use environment is needed in order to be able to introduce new services as desired.

The customers of a dynamic VPN service will include not only corporate users who currently employ VPNs over the internet or leased lines for their private network, but also companies selling network services themselves. These might be telephone companies, ISPs, off-the-street users needing a network for a short time, perhaps for a distributed game or a broadcast lecture, or even designers and developers of other network services. All these customers will have different needs over different timescales—some VPNs will be semi-permanent, while others rapidly and frequently change their resource requirements, or even only persist for a matter of minutes.

To fully realise the potential of the technology in this scenario, the VSP must offer a responsive, reliable and flexible service. Administrative overheads must be minimised; in particular the automated creation and modification of VPNs is paramount, not only to cope with large volumes of such requests, but also to be able to respond to them quickly and to cater for the many different types of customer. The remainder of this paper considers VPN specification and realisation in this context.

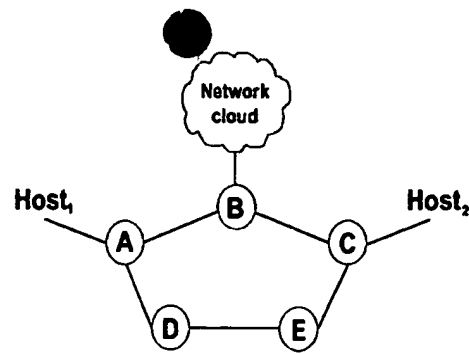


Fig. 3. Example network where preferred VPN topology from Host₁ to Host₂ is different for VSP and customer.

III. VPN DESIGN

So far this paper has described programmable VPNs and their advantages, as well as reviewing the switchlets concept and the prototype implementation of a VPN Service that provided the motivation for this work. The deployment of such a service on a real-world network is an attractive prospect, but in order to cater to the demands created by short-lived, lightweight and dynamic VPNs, automation of their provisioning is essential. The challenges of doing so are now investigated, with the overall aim of showing that in spite of some inherent difficulties, the automated design and deployment of VPNs is a feasible procedure.

The goal in VPN design is to create a virtual network that conforms to the specification provided by the customer, whilst maximising subsequent resource availability for the VSP. In order to increase the likelihood of being able to satisfy future customer requirements, the VSP should endeavour to spread the load of bandwidth, label space and switchlet allocation. As an example, Fig. 3 shows a network topology where although a customer might prefer that their VPN from Host 1 to Host 2 passes through the single node B, the VSP will route the VPN through D and E in order to maximise the local resource availability for VPNs originating within the network cloud.

This section considers the two main steps of the design process—customer specification of the desired VPN, and a means of mapping that specification to a near optimal topology and resource allocation.

A. VPN Description

The characteristics of a VPN that a customer may specify in a creation or modification request include:

- (virtual) topology;
- performance characteristics such as bandwidth, delay, loss tolerance, size of label space etc;
- temporal attributes i.e. start time and duration;
- cost;
- built-in extras, such as redundancy;
- contractual issues—penalties etc.

This type of specification, which describes both the desired VPN and specifies the guarantees made by the VSP with respect to that VPN, is often known as a *service level agreement* (SLA). Traditional SLAs used by WAN service providers are contracts between customer and network service provider that detail the level of service agreed in terms of measurable parameters. The content of an SLA usually covers type of service, data rate and QoS issues as well as contractual matters such as charging and compensation in the event of non-compliance with the agreement.

The automated translation of a VPN specification to a set of physical resources requires a formalised notation that is sufficiently expressive to capture a range of requirements, but is not unnecessarily restrictive. As pointed out in [8], VPN descriptions may range from the ill-defined (eg, "a cheap network between A and B"), to the comprehensively specified. Furthermore, the difficulties of predicting network traffic characteristics from a given source are well known. Insisting on a precise specification when the user does not know or understand their requirements in such detail only leads to sub-optimal network usage, either because the source exceeds its usage parameters and loses data, or else because the network is over-provisioned and resources wasted.

A VPN may not necessarily be double-ended, in other words a VPN specification which describes the origin and characteristics of the traffic without explicitly stating its destination should be valid. The conversion to a complete specification should be able to take into account potential optimisations resulting from this single-endedness to multiplex where possible.

The exact nature of the notation adopted depends to some extent on the capabilities of the underlying network and the degree of specification that the VSP wishes to allow its customers. Greater freedom leads inevitably to more complexity in the system. In our experimental environment we allow a fairly limited choice of VPN parameters, namely participating nodes, size of label space, bandwidth and maximum hop count. Other characteristics that could easily be incorporated include delay, jitter, duration, start-time, redundancy and so on.

After an SLA is defined, the partial VPN specification it contains must be converted to a complete specification, that is, one that represents an instantiation of the specified VPN on the actual physical networks. This involves an augmentation of the incomplete description such that all of the necessary physical topology, together with the resources required on each node and link, is explicitly specified.

Once a complete description is arrived at, a VPN can easily be expressed as a set of switchlet specifications. The mapping of a customer-provided VPN description to a complete description is the most complex step in the process of creating a VPN. Two questions must be addressed:

1. Does the described VPN make sense in terms of the actual physical topology—is the description *feasible*?
2. How do we arrive at an arrangement of VPN topology which:
 - satisfies the customer requirements,

- can be realised with available resources, and
- is optimal for the VSP?

Checking of feasibility is straightforward, as the VSP has global knowledge about the network topology. At this step it may also be able to refine the SLA so that it includes at least those nodes that *must* be present to meet the stated requirements. In contrast, the second question is quite difficult to answer, and is discussed at length in the next section.

B. VPN Routing

Determining an optimal route, or topology, of a VPN is a non-trivial problem. It can be expressed formally as follows:

The cost of a subgraph G' is determined by some function $f(G')$. Given a weighted graph G and a set of vertices in G denoted V , what is the cheapest way of forming a connected subgraph containing V according to the cost function f ?

This problem is similar to that of finding a minimum Steiner Tree, where the goal is to connect a set of vertices in the graph by finding a minimum-weight spanning tree that can also use any of the remaining vertices. The difference is that the aim is not to find a minimum spanning tree, but to find the cheapest subgraph according to a cost function that will vary from one VSP to another, and may even be altered over time at the same VSP.

As an example consider the networks shown in Fig. 4. The requested VPN consists of the nodes *A*, *B* and *C*, and the edge weights have been calculated as shown, according to the VPN resource description, current resource availability and local policy. A costing function that favours the minimum-weight VPN topology will produce the subgraph highlighted in the left-hand network, whereas the subgraph of the middle network is produced if minimising the number of links is given greater priority. If full redundancy together with minimum-weight is required, the subgraph highlighted in the right-hand network will be the result.

It is clear from the problem description that once a partial VPN specification has been derived from the given SLA, there are then three stages in reaching a solution:

B.1 Assign edge weights

The weight assigned to any individual edge will depend on:

- availability of the resources specified in the VPN description (either explicitly or implicitly) as required for that link;
- any arbitrary cost associated with a link, as determined by local policy.

A link that is unable to provide the required resources is immediately assigned a weight of infinity and removed from further consideration.

B.2 Generate a cost function

The cost of a VPN is determined by a combination of customer requirements—a candidate VPN that doesn't meet the specifications of the SLA will have infinite cost—and the circumstances of the individual VSP. For example, in some cases bandwidth may "cost" more than label space (perhaps because

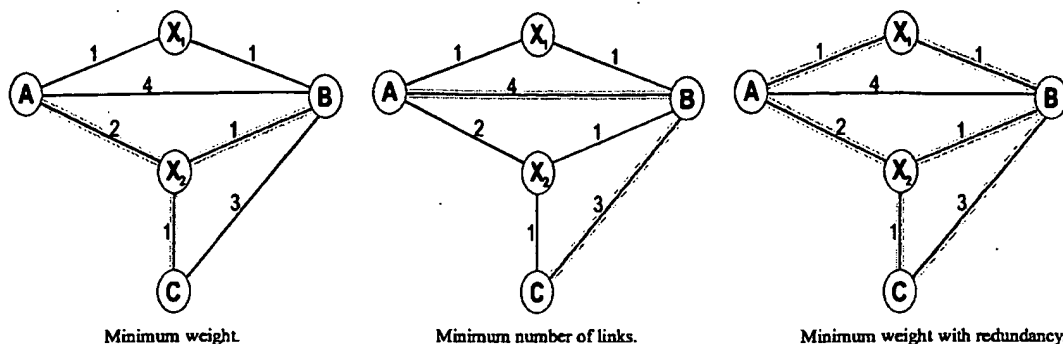


Fig. 4. Three cost functions producing different optimal VPN topologies.

the VSP has very few customers), whereas in other networks the reverse will be true. The costing function should reflect the relative priorities in the local domain, and balance the requirements for a particular VPN with the overall needs of the VSP.

A cost function routine is automatically generated by parsing the SLA to derive explicit rules and incorporating these in a template that includes rules dictated by local policy. The details of this procedure will vary from one VSP to another, but there is always commonality in that the cost calculation routine used at any one time is (probably) unique to the individual VPN under consideration.

B.3 Find the best VPN topology

As stated above, a VPN's topology will be determined by a combination of the edge weights and the cost function, according to local policy.

Intuitively, an implementation that always finds the best subgraph (i.e. VPN topology) will be computationally expensive because of the large number of candidate subgraphs. If we consider a subgraph cost function that is simply the sum of the constituent edge weights, and if cycles are not permitted in the subgraph, then the problem becomes that of finding the minimum Steiner Tree. This is known to be NP-complete [15], and thus so is the optimal VPN topology problem described here.

Nevertheless, using a combination of sensible heuristics and careful engineering, an implementation can be produced that gives tolerable performance for a network of reasonable size. In the following section the brute-force solution is analysed in order to derive heuristics that give close to optimal solutions, as well as making the problem tractable within the desired bounds. These bounds are determined by deciding what "tolerable" performance is, and how large a "reasonable" size of network should be. Experimental observation supports the claim that the performance of the resulting algorithm is adequate.

IV. IMPLEMENTATION EXPERIENCE

This section examines in some detail the implementation of the search for an optimal VPN topology. By means of experimental results we compare the performance of the naive approach with that possible using a simple heuristic.

A. Algorithm

A brute force method of determining the optimal VPN topology is to construct every possible connected subgraph that satisfies the given VPN description, calculate the cost of each and then choose the cheapest. The pseudo-code of Fig. 5 describes an algorithm that takes this approach. For clarity, finer points of the implementation such as finding optimal topologies that incorporate redundant links are ignored.

Let V' denote the set of nodes in the VPN. For any $v \in V'$, $\text{single-hops}[v]$ contains all paths from v to each of the other nodes in V' , that do not pass through any other node in V' and do not contain any cycles. The function CALCULATE-COST is generated as described in Section III-B.2.

The construction of single-hops is straightforward using a modified form of the Floyd-Warshall transitive closure algorithm to generate one possible set of paths between VPN nodes, and then taking the transitive closure of the resulting paths to obtain *all* possible paths. At this stage paths with edges that have infinite weight (i.e. cannot fulfill the resource requirements) are discarded.

The algorithm to find the minimum-cost subgraph containing V' is shown in Fig. 5. At line 7 whichever VPN node was added last to the subgraph is taken, and its paths to other VPN nodes looked up in single-hops . The procedure is then called recursively for each of these paths. Note that the resulting collection of paths will not necessarily be disjoint, but a subgraph is a candidate (i.e. costed) only if it contains all of the nodes in the VPN. At line 8 processing is also carried out to ensure that continuously cycling paths are not considered, and to guarantee termination within a reasonable time period, the search is abandoned after a certain (large) number of recursions.

```

MIN-SUBGRAPH(subgraph)
1  if all nodes  $V'$  in subgraph then
2       $c = \text{CALCULATE-COST}(\text{subgraph})$ 
3      if  $c < \text{min-cost}$  then
4           $\text{min-cost} = c$ 
5           $\text{min-subgraph} = \text{subgraph}$ 
6  else
7       $\text{paths} = \text{single-hops}[\text{subgraph.last}]$ 
8      for each  $p$  in paths do
9          MIN-SUBGRAPH(subgraph +  $p$ )

```

Fig. 5. Brute-force algorithm to find a minimum-cost subgraph.

Let m be the number of edges in the graph G and let k denote the number of nodes in the VPN. The number of paths between any two nodes without cycles is at most 2^m , and for any node v there are paths to at most $k - 1$ other nodes. Therefore the maximum number of iterations of the loop at line 8 is at most $(k - 1)$, with each iteration recursing at most 2^m times. As expected, the brute-force algorithm is computationally intractable.

B. Heuristics

The intractability of this algorithm arises because the cost function is not monotonically increasing, i.e. the overall cost of a subgraph may reduce as it encompasses more of the VPN nodes (for example if redundancy is required by the customer). As stated above, if the cost function does happen to be cumulative, then the solution is the minimum spanning tree and can be found in polynomial time using any of the well known algorithms.

However, even with a non-monotonic cost function, heuristics can be applied to make the procedure practical for networks of reasonable size. At the potential expense of sub-optimal results, the search time can be speeded up by reducing the size of the search space. Two approaches are possible:

- abandon partial subgraphs that are thought likely not to be optimal in the future;
- order the search sequence to consider first those subgraphs that are most likely to be optimal and stop the search at the first hit.

The first approach can be implemented by choosing a reasonable cut-off point, and abandoning partial subgraphs that exceed this cost. This is obviously more likely to perform well if the cost function is 'almost' monotonic. Additionally, the cost function may contain cumulative aspects that can be considered in isolation. For example, partial subgraphs can be discarded on the basis of exceeding the desired hop count or end-to-end delay. However, a significant drawback with this approach is that the additional computation associated with calculating costs of partial subgraphs may dominate the running time and negate any improvements gained from the smaller number

of subgraphs considered in total.

In contrast, the second technique will tend not to return as good results, especially with denser graphs, but will spend significantly less time calculating partial subgraph costs. Another key advantage is that if a subgraph that meets the customer's requirements (with the possible exception of a cost constraint) exists, then it will always be found. The first technique runs the risk of not finding a solution, due to having abandoned that subgraph earlier, whereas with ordered search if a solution exists then it is guaranteed to be found eventually.

Thus with the ordered search heuristic the VSP may lose out on fulfilling its overall goals, and the customer may be penalised on VPN price, but both gain substantially from search speed increases. Of course in practice the VSP can subsequently mitigate any losses by modifying the cost function appropriately. The results in the following section support this analysis, and demonstrate that by incorporating the ordered search heuristic, automated VPN design is a feasible procedure.

C. Experimentation

This section presents experimental results to back up the assertion that in spite of its computational complexity, the process of automated VPN creation is practical on a reasonably large network.

For ease of implementation, an interpreted scripting language was used, hence the CPU usage timings give an idea of relative improvements rather than demonstrating what can be achieved absolutely. A well-engineered solution written with the appropriate tools would perform much better. The network topologies were generated using the TIERS random network topology simulator [16], with a small range of edge densities reflecting the sparsity generally found in real-life networks. The networks are assumed to be under the administrative control of a single VSP and are accordingly no larger than 25 nodes.

Because the topologies are produced randomly no two runs give the same results and there are occasional large fluctuations for relatively large and dense networks. Notwithstanding this, some care has been taken to ensure that the results included represent typical executions, and all data points are average values over 10 runs.

The graph in Fig. 6 shows how badly the brute-force algorithm actually performs in practice. It indicates the extreme deterioration in performance as both the graph size and the proportional VPN size increase. Once the VPN covers more than about 30% of the network, the search time increases dramatically.

Problems are also caused for brute-force searching by increases in graph density, and this is shown by the graph in Fig. 7. This graph shows the search time for a VPN covering a fixed proportion of the physical network—60%, with average node degree increasing from 1 to 4. Search times increase substantially once the graph size exceeds 10 nodes. Clearly the brute-force approach is not adequate for any network of reasonable size.

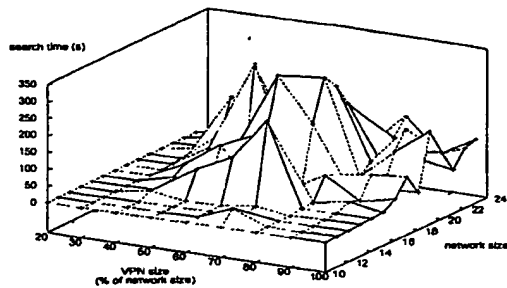


Fig. 6. Performance of brute-force search on sparse networks.

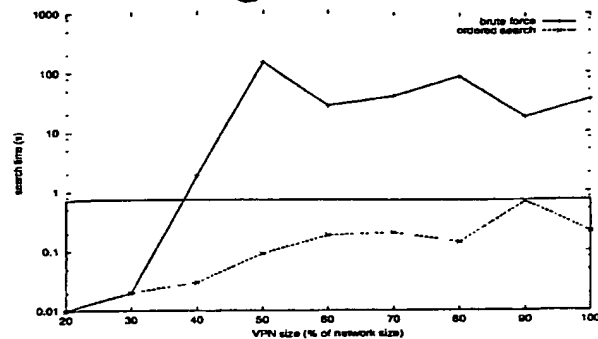


Fig. 8. Brute-force vs ordered search on a 20-node sparse network.

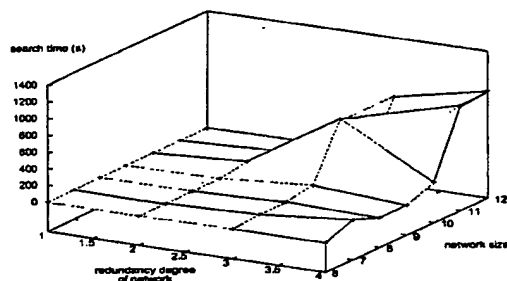


Fig. 7. Performance of brute-force search for VPN size 60% of network size.

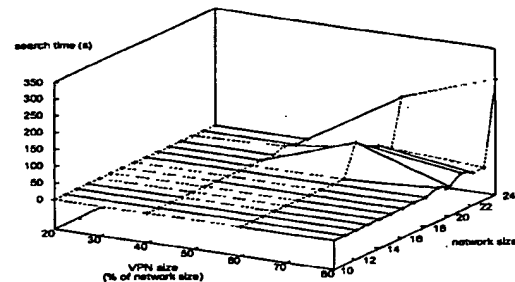


Fig. 9. Performance of the ordered search heuristic on sparse networks.

The next experiment examines the gains that can be made using the ordered search heuristic as well as the corresponding cost penalty. The heuristic is incorporated in the pseudo-code algorithm of Fig. 5 by ordering the paths list iterated over at line 8 according to the result of applying the cost function to each one. Recall that these are acyclic single-hop paths between 2 nodes of the VPN, therefore the overheads of the cost calculations are minimal.

The graph in Fig. 8 compares the search times of the brute-force algorithm and the ordered search algorithm for a medium-sized graph of 20 nodes. It shows the expected marked improvement in performance using the heuristic. Note that the y-axis is log scale.

Finally, the first two experiments run using brute-force searching are repeated with the ordered search heuristic. To facilitate comparison, the results are plotted with the same Z-axis range which confirms that the large humps present in the graphs of Figs. 6 and 7 only appear here for much bigger networks. As expected, ordered search performs much better in both cases.

V. FURTHER WORK

So far the issue of automated VPN creation has been addressed without regard to possible reconfigurations of the VPN in the future. In fact, the ability to alter the topology and resource allocation of a VPN on-the-fly is one of the main advantages of the VPN Service described in this paper. Some scenarios where dynamic reconfiguration might be used include:

- A service-specific control system tailored to a particular multicast application, for example video-conferencing, where changes to group membership may require switchlets to be created at additional nodes, or even for nodes to be removed from the VPN. Similarly, end-user requests for improvements in video quality may be met by increasing the bandwidth allocation of the VPN on the relevant links.
- A control system for a VPN supporting mobility that modifies its topology in response to the movement of wireless devices—holding on to resources only at adjacent base stations, and discarding those that are “far away” from the current location.
- An efficient IP-on-ATM control system where the size of the label space in the underlying switchlets is altered rapidly in response to the creation or termination of traffic flows.

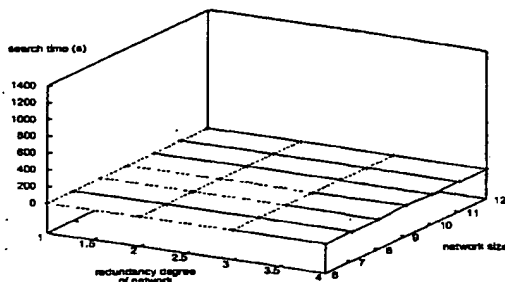


Fig. 10. Performance of the ordered search heuristic for VPN size 60% of network size.

VPNs that reconfigure as and when required can make more efficient use of network resources and are more flexible than statically pre-configured networks. The risk for the VPN owner is that a reconfiguration request may be rejected by the VSP, perhaps because of lack of resources, with the resulting loss for the VPN owner of efficiency, flexibility and possibly even the ability to provide a service to their own customers. The appropriate charging mechanisms to reflect this trade-off are the responsibility of the VSP.

Automated reconfiguration should be built in to the system for the same reasons that automated creation and deletion are necessary, as discussed in Section II-C. Reconfiguration can involve changes to either the VPN topology, or to its resource allocation, or both. The problem considered here concerns just alterations to VPN topology, which may of course result in changes to the overall VPN resource allocation as well. The details of in-place reconfiguration of resources at a particular switch are currently the subject of ongoing research.

A. Analogy With Dynamic Multicast Routing

The problems of dynamic VPN topology reconfiguration parallel those found in dynamic multicast routing (but not for datagram networks). The issues, which are extensively discussed in the literature (see, for example [17]) involve compromises between optimal placement of a node joining the multicast session and the corresponding deterioration of the tree as a whole.

As with VPN creation, the initiation of a multicast session generally involves a computation of the optimal topology. The topology of multicast trees, which are single source to multiple receivers, can be influenced by many different constraints. These include scalability, QoS requirements such as end-to-end delay bounds, efficiency requirements and algorithm complexity considerations. When a new destination is introduced into the multicast tree, naive (computationally cheap) placement of the new node can result in deterioration of the quality of the tree—with some chance of making the initial effort expended in calculating an optimal tree a waste of time. On the other

hand, constant reconfigurations of the topology to maintain an optimal, or close to optimal, tree are also expensive, and can disrupt traffic to existing members of the multicast group. Most solutions adopt a compromise where changes to the tree are kept as localised as possible, and periodically the entire tree is re-routed to try and maintain a close to optimal topology.

Some examples of current research in this area include [18] and [19]. In [18], an algorithm is presented that maintains a good (but not optimal) multicast tree (in terms of minimising the sum of the edge weights) without undue computational cost using Kruskal's shortest-path algorithm. Rearrangements are triggered after a certain amount of deterioration in the quality of the tree—determined by counting the number of changes in a vicinity—but enough state is maintained to be able to confine the necessary rearrangements to localised regions. In contrast, in [19] the quality of the multicast tree is assessed according to whether it meets delay variation constraints. However there is a similar emphasis on minimising the effects of leave and join operations on the tree as a whole.

B. Discussion

Although it has a lot in common with the multicast problem, dynamic reconfiguration in the context of VPNs in the switchlets environment has some important differences.

On the one hand, the allocation of physical network resources directly to a VPN owner means that topological rearrangements of the VPN at the whim of the VSP are not necessarily going to suit the way the customer is using their portion of the network. On the other, the difficulties can be eased by the fact that the customer may be able to explicitly identify "sensitive" regions of their network that should not be modified, and nodes where disruption is tolerated. Any threshold of deterioration (which will roughly translate to cost to the customer) can be chosen on a per-VPN basis, and to take this to an extreme, the mechanism by which the topology is updated can also be specified by the VPN customer. In effect this gives the VPN owner, i.e. the person paying for the network, a great deal of control, allowing them to tailor the dynamic reconfiguration behaviour as appropriate.

However, it must also be expected that many VPN customers will not be willing or able to specify such details. A minimal VPN specification such as "Give me a network containing nodes A, B and C" which at some point in the future is modified by "Add node D to my network" is a perfectly valid one. The question is whether this customer would be happy with a less than optimal topology where D is simply joined to A, B and C, or whether a rearrangement of all the links in the VPN at this point would be acceptable. In general it seems unlikely that the latter option would be preferred, and indeed should a customer require this they could always request a new network containing A, B, C and D at the appropriate time. On the other side of the coin, the VSP may have expended much effort in calculating the original network topology, and this effort may subsequently be rendered useless by the changes requested by the customer, especially if these changes occur frequently. It would be in the interests of

the VSP to have some sort of characterisation of the "dynamicity" of a VPN at the time of creation. A VPN that is likely to change a lot over its lifetime can be given a topology that is sub-optimal in terms of cost but is more resilient and gives a cheaper overall topology with plentiful changes.

In summary, the following aspects relating to reconfiguration can be specified by the VPN customer:

- the expected rate of membership modification;
- tolerance of disruption to the VPN as a whole;
- tolerance of disruption at particular nodes;
- any preferred means of rearrangement;
- thresholds for triggering rearrangement;
- amount prepared to pay.

A combination of these factors could be used to influence the choice of initial routing. A VPN that underestimates its degree of dynamicity will initially pay an extra cost for unanticipated reorganisation overheads. However it is also possible in this situation that the system could slowly adapt to the observed behaviour of a misbehaving VPN, and successively produce less optimal but more resilient topologies.

The nature of a VPN topology that is resilient to change, in terms of maintaining its overall cost as its membership alters, will vary according to the cost function in use by the VSP. Algorithms proposed for dynamic multicast routing, such as ARIES [18], could be adapted to operate in this environment according to the chosen cost function. Investigation into this aspect of the provision of dynamic VPNs is continuing.

VI. CONCLUSION

The automation of VPN design and deployment has many advantages for both VSP and customer. It allows a customer to obtain a VPN extremely quickly, while still being able to explicitly specify the VPN's performance and other characteristics. The administrative overheads for the VSP are greatly reduced, and network usage efficiency is enhanced. With the incorporation of charging mechanisms correlated against computed VPN cost, the administrative burden can easily be further minimised.

This paper has shown that in spite of computational complexities it is feasible to automate VPN topology generation in accordance with both customer requirements for the VPN itself and VSP goals for the network as a whole. A VPN service level agreement, together with current resource allocation and global policy, can be incorporated in a cost function that is then used to determine the optimal physical topology of a VPN that satisfies the specification. The generation of this topology is made computationally tractable by means of an ordered search heuristic. Experiments have demonstrated that even within a non-optimised environment, a VPN can generally be determined in under 2 minutes on a sparse, reasonably sized network of up to about 25 nodes.

A programmable network infrastructure, such as that facilitated by switchlets, opens up the way in which a VPN can be exploited. As well as a flexible and dynamic network topology and resource allocation, there are no built-in restrictions on the

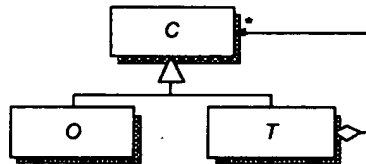
control system, network protocol or end-user applications using that network. This "open" environment, with lightweight and dynamic VPNs, is a practical realisation of "networks-on-demand".

REFERENCES

- [1] S.Blake, D.Black, M.Carlson, E.Davies, Z.Wang, and W.Weiss. An architecture for differentiated services. RFC 2475, December 1998.
- [2] Jacobus E. van der Merwe and Ian Leslie. Switchlets and dynamic virtual ATM networks. In Aurel Lazar, Roberto Saracco, and Rolf Stadler, editors, *Integrated Network Management V*, pages 355-368. IFIP & IEEE, Chapman & Hall, May 1997.
- [3] Opensig working group. Details at <http://comet.columbia.edu/opensig/>.
- [4] Jacobus E. van der Merwe and Ian M. Leslie. Service specific control architectures for ATM. *IEEE Journal on Selected Areas in Communication*, 16(3):424-436, April 1998.
- [5] P. Newman, W. Edwards, R. Hinden, E. Hoffman, F. Ching, T. Lyon, and G. Minshall. Ipsilon's General Switch Management Protocol specification version 2.0. RFC 2297, March 1998.
- [6] William P. Buckley. Virtual Switch Interface (VSI) implementation agreement. Available from <http://www.msforum.org/>, November 1998.
- [7] Sean Rooney, Jacobus E. van der Merwe, Simon A. Crosby, and Ian M. Leslie. The Tempest: A framework for safe, resource-assured programmable networks. *IEEE Communications Magazine*, 36(10):42-53, October 1998.
- [8] Jacobus E. van der Merwe, Sean Rooney, Ian Leslie, and Simon Crosby. The Tempest—a practical framework for network programmability. *IEEE Network Magazine*, 12(3):20-28, May 1998.
- [9] Ian Leslie, Derek McAuley, Richard Black, Timothy Roscoe, Paul Barham, David Evers, Robin Fairbairns, and Eoin Hyden. The design and implementation of an operating system to support distributed multimedia applications. *IEEE Journal on Selected Areas in Communication*, 14(7):1280-1297, September 1996.
- [10] Herbert Bos. Application-specific policies: Beyond the domain boundaries. In Morris Sloman, Subrata Mazumdar, and Emil Lupu, editors, *Integrated Network Management VI*, pages 827-840, Boston, May 1999. IFIP & IEEE, Chapman & Hall.
- [11] David L. Temmenhouse, Jonathan M. Smith, W. David Sincoskie, David J. Wetherall, and Gary J. Minden. A survey of active network research. *IEEE Communications Magazine*, 35(1):80-86, January 1997.
- [12] Andrew T. Campbell, Michael E. Koumavis, Daniel A. Vilella, John B. Vicente, Hermann G. De Meer, Kazuho Miki, and Kalai S. Kalaichevan. Spawning networks. *IEEE Network Magazine*, 13(4):16-29, July/August 1999.
- [13] Andrew Do-Sung Jun and Alberto Leon-Garcia. Virtual network resources management: A divide-and-conquer approach for the control of future networks. In *Proceedings of the IEEE Global Telecommunications Conference (GlobeCom 98)*, Sydney, Australia, 1998.
- [14] N.G. Duffield, Pawan Goyal, Albert Greenberg, Partho Mishra, K.K. Ramakrishnan, and Jacobus E. Van der Merwe. A flexible model for resource management in virtual private networks. *Computer Communication Review*, 29(4):95-108, October 1999. Proceedings of SIGCOMM September 1999.
- [15] Michael R. Garey and David S. Johnson. *Computers and Intractability: A Guide to the Theory of NP-Completeness*. W.H. Freeman and Company, 1979.
- [16] Matthew B. Doar. A better model for generating test networks. In *Proceedings of the IEEE Global Telecommunications Conference (GlobeCom 96)*, pages 86-93, London, UK, November 1996. Source code available from <ftp://ftp.nexen.com/pub/papers/tiers1.2.tar.gz>.
- [17] Matthew Doar and Ian Leslie. How bad is naive multicast routing? In *IEEE INFOCOM'93*, pages 82-89, San Francisco, USA, April 1993.
- [18] Fred Bauer and Anujan Varma. ARIES: A rearrangeable inexpensive edge-based on-line Steiner algorithm. *IEEE Journal on Selected Areas in Communication*, 15(3):382-397, April 1998.
- [19] George N. Rouskas and Ilia Baldine. Multicast routing with end-to-end delay and delay variation constraints. *IEEE Journal on Selected Areas in Communication*, 15(3):346-356, April 1998.

Best Available Copy

***Experiences from extending a legacy
system with CORBA components***
COT/4-08-V1.3



Centre for Object Technology

*Centre for
Object Technology*

Revision history: V1.0 1999-03-01 First draft.
 V1.1 1999-05-18 Second draft to review.
 V1.2 1999-06-21 Third draft to review.
 V1.3 1999-10-01 Final version.

Author(s): Allan R. Lassen, RAMBØLL
 Jacob Steen Due, RAMBØLL
 Emil Hahn Pedersen, RAMBØLL
 Mohammad Al-Shamri, RAMBØLL

Status: Final

Publication: Public

Summary:

A presentation of the experiences gained by performing an extension of a legacy system by using object-oriented technology. The experiences are the basis of a general approach to extend a legacy system with additional functionality based on the CORBA technology.

© Copyright 1999

The Centre for Object Technology (COT) is a three year project concerned with research, application and implementation of object technology in Danish companies. The project is financially supported by The Danish National Centre for IT-Research (CIT) and the Danish Ministry of Industry.

Participants are:
Maersk Line, Maersk Training Centre, Bang & Olufsen, WM-data, Rambøll, Danfoss, Systematic Software Engineering, Odense Steel Shipyard, A.P. Møller, University of Aarhus, Odense University, University of Copenhagen, Danish Technological Institute and Danish Maritime Institute

1. Introduction

In the IT industry many systems, developed a long time ago, are still maintained because they are crucial for the IT business. Introducing new and better technology in the development process raises a challenge on how to adopt the new technology in the older systems. In this document we present our experiences of extending a system using object-oriented technology and CORBA. For more information on our project please refer to [3] which contains the design of the new system.

2. Goal of project

The goals of this project were first of all to gain experience and secondly to produce a final product. The main factors were:

- We saw a need to develop a method that allows new functionality to be added to older legacy¹ systems.
 - The new functionality should be in the form of add-on modules.
 - The changes needed in the legacy system should be minimised (and ideally there should be no changes at all in the legacy system).
- The architecture should allow implementation in multi-tier architectures.
 - We wanted to implement it in CORBA using the latest version of Oracle Application Server (OAS). This was basically a commercial reason. We had good experience with other Oracle tools and we expected that OAS would integrate nicely with those Oracle databases that many of our systems use.

The project is part of the research conducted in the Centre for Object Technology [1]. The actual project is part of industrial case 4 concerning integration with non-OO systems. By using object-oriented technology a partial goal of the project was to examine if the above-mentioned requirements could be realised successfully.

The basis of the project was a workflow and document management system, WorkSAFE, developed by RAMBØLL [2] as a standard system. Although the system is relatively new and well documented it corresponds to the definition of legacy system used in this report.

The current workflow system is designed to be used internally in an organisation. We wanted to give external partners access to selected parts of the system (i.e. access to some of the documents and to participate in the workflow for quality assurance). Partners should learn the system quickly. Eventhough the current system has a very general user interface, it takes time to learn and we wanted to supply a user interface designed to meet the requirements of the partners.

¹ In this report we use the term 'Legacy system' to denote monolithic systems. Typical examples are custom developed systems for various purposes such as accounting and other administrative purposes. Such systems are traditionally difficult and expensive to develop and maintain.

Whereas this may seem as a rather trivial case it does in fact raise a number of interesting questions that have turned out to produce implications in general when adding new functionality to legacy systems:

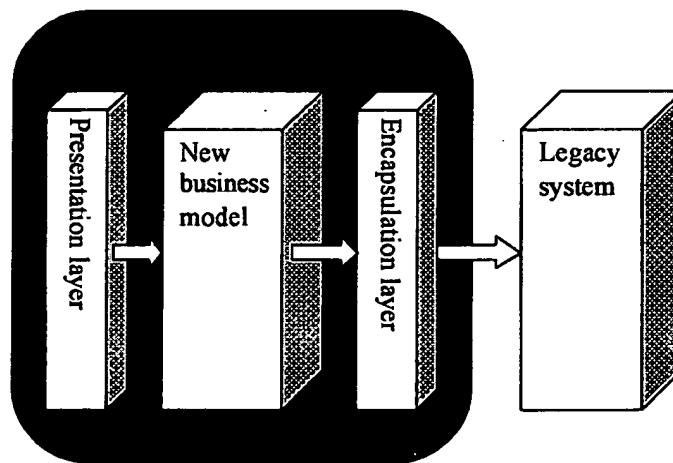
- How do we get a maintainable interface to the legacy system?
- How do we reduce our dependency on changes in the legacy system?
- How does the choice of technology influence our implementation?

3. Architecture

A key issue in adding new functionality to the legacy system, is the architecture to be used for the add-on modules. You may ask yourself the question: What kinds of interfaces are available for the legacy system?

Many legacy systems are using a relational database as persistent storage. The database tables implement an entity-relationship model, and usually you can easily transform it into an object model by representing each entity as a class. However, in our case the system had a complicated E/R-model because it used a meta-model of all its data. Therefore we decided to use an interface based on its Oracle database views and stored procedures. In another system you may have other alternatives, such as a documented API.

In our project we came up with a logical architecture, which we in general find useful. The architecture is illustrated in the figure below.



The encapsulation layer contains the classes corresponding directly to the legacy system interface. The layer is (and must be) as simple as possible and contains only the functionality that exists in the legacy system. The purpose of this layer is to shield the new business model from the complexities and idiosyncrasies in the actual implementation of the legacy system. Therefore it is very important to make the layer as simple as possible.

The new business model is a model of the relevant parts of the legacy system and contains classes with business logic of the new functionality. The classes use the encapsulation layer to perform actions towards the legacy system. The business classes serve as an abstraction of the legacy system for the user interface and they only contain information that is relevant to the new clients.

In a traditional multi-tier architecture there is a database tier and one or more business application tiers. In our architecture the business logic is performed in a combination of the new business model and the legacy system.

The presentation layer implements the user interface like in traditional multi-tier architectures.

3.1 Considerations about the clients

In our project the new functionality did not depend on the existing user interface and we considered three solutions to the presentation layer:

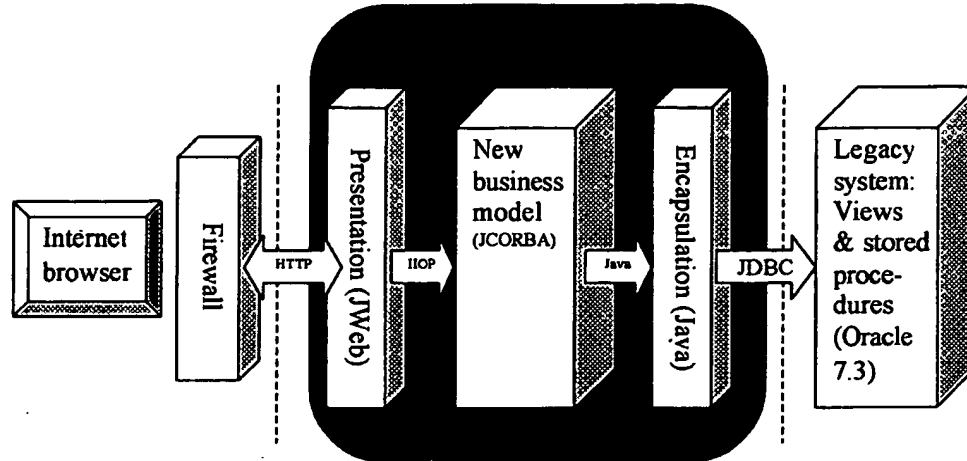
- An HTML based presentation where HTML pages are generated from the presentation layer on a server and displayed in an Internet browser on the client.
- A presentation using Java applets running within the Internet browser. The applets are Java classes that will be loaded from a server by the browser and then run on the client. The applets will communicate through CORBA IIOP with business classes running on a server.
- A stand-alone client application e.g. developed in Java. The application does not require a browser and will communicate with business classes in the same way as Java applets. Some mechanism to install the application would be needed.

We believe the Java applets or application would offer the best facilities for implementing a nice and fancy user interface. However to have our end-users, i.e. a partner, accessing the system through the Internet raises a problem with firewalls. Currently most firewalls do not allow the CORBA IIOP protocol to pass through. Eventhough, we get our company firewall to work with IIOP the user will probably sit behind a partner firewall that we not are able to control. Therefore we decided to dynamically generate HTML from our presentation layer.

An alternative is to implement a Java based user interface by marshalling and tunnelling the Java objects through the HTTP protocol. But it requires some extra work to implement and we didn't want to struggle with this issue in our project. Hopefully the problem about getting the IIOP protocol through firewalls will soon be solved and thus make implementation of fancy user interfaces easier.

3.2 Used technology

The Oracle Application Server 4.0 is the platform of our implementation of the architecture and we use techniques from the product as illustrated below:



The figure shows a physical model of our new system. All the classes in the system are implemented in Java. The business classes are subclasses of a JCORBA superclass, which is a class that Oracle has derived from the standard OMG CORBA class. JWeb classes are classes that dynamically generate HTML pages from Java. The dotted, vertical lines separate the parts that can be placed on separate physical servers.

4. Development process

4.1 Modelling the problem domain

The architecture described in the previous section naturally leads to the following steps in the modelling.

- An external model of the legacy system must be created in order to provide the encapsulation layer.
- A new business model is made.
- The encapsulation and business models are mapped together.

In our case it was relatively easy to construct the encapsulation layer because of the interfaces available to the legacy system, but this is not always the case. In other systems the nature and structure of the legacy system may make the task more complicated, e.g. if the only way to enter and query information in the legacy system is through its user interface. Though the goal is to use existing code from the legacy system there is a risk on having to establish some of the functionality in the encapsulation layer.

The business model could be created with only little awareness of the old legacy system and with focus on the application field. In effect this model was created in the same way as creating a model for a completely new system. Of course the model has a constraint that it must conform to the legacy model.

Interestingly enough, we found out that eventhough the developers with real knowledge of the legacy system found it relatively easy to create the encapsulation layer, they also found it demanding to create the business model due to the difficulty of abstracting from the familiar terms used in the legacy system. Actually, the participants with superficial knowledge of the legacy system were the ones who made the greatest progress on the business model. In practice two tracks were formed to develop the layers in parallel and this worked surprisingly well because of the clear division of the responsibilities in the layers.

In our case the business model did not introduce the need for extra information - it could all be stored in the legacy system. Therefore the problems associated to such a need have not been an issue. If such a need occurs the correct place for storing the information is in the business model when it is required that the legacy system shall not be modified. However, the complexities in securing transactional control, data security and performance are not described in this project, eventually an exercise for the reader ☺.

The objects in the business model may be distributed when they are implemented as CORBA objects. This raises an interesting opportunity to obtain better fault tolerance in the system. However we didn't investigate this topic further in our project.

The mapping between the encapsulation and business models proved to be relatively easy due to several reasons. One reason is the major factor that the problem domain for the new functionality was close to the problem domain of the old legacy system. This must be assumed to be the case in most situations where add-on functionality is put on top of legacy systems. Honestly, our legacy system was developed with OO techniques, as an implementation of an OO model in the relational paradigm. Though this implementation probably reduced the complexity in mapping the models we still believe our experience is correct because we interface to the legacy system as a traditional system implemented in a relational database.

4.2 Functionality

In developing the object structure of the business model and methods we had good experiences with small descriptions of scenarios². The scenarios illustrated the work processes we wanted to support and they were combined with the involved dialog flows. The scenarios and dialog flows helped to identify key methods and events in the system.

² The scenarios descriptions could have been replaced by the standardised use cases from UML but we chose to use descriptions that were less formal.

4.3 Design

After creating a good business model we found it difficult to describe a detailed design without real knowledge on the implementation platform. It is a general challenge when you take new technology in use. We believe it is a good idea to try some experiments and perhaps find some examples to benefit from. Our project described in [3] may serve as an example. It is also a good idea to see if there is a design pattern describing a solution to similar problems. An introduction to patterns is given in [4].

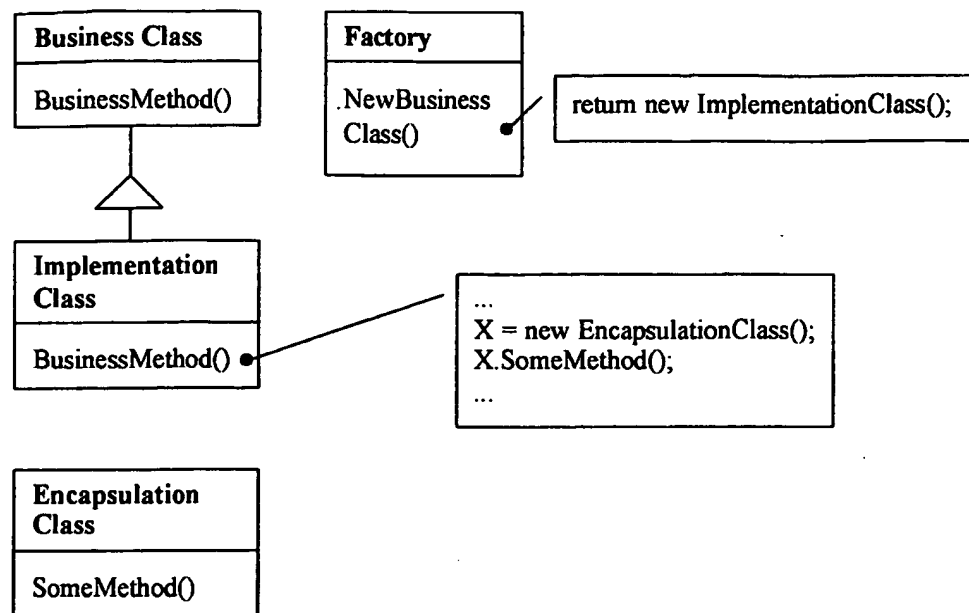
One of our concerns was how the clients should connect to the system and how we could keep track of the connection between user interactions. We found a good solution by introducing a session class and a session manager that handles the user connections (see [3] for details).

Our object-oriented system uses the relational database in the legacy system as persistent storage. It raises a question on how to manage the information when it is represented both in the relational database itself and in the object instances that we create in our system. We wanted to be certain that the information was updated correctly in our system as well as in the legacy system. Indeed, it is a problem if the legacy system changes the object values after our system have loaded them into an object instance or vice versa. We were lucky that the legacy system had implemented a locking strategy we could use to notify when we were going to update the information. In another system you may have to struggle with this issue.

In order to handle the information in the legacy system we used the following two general principles:

- We introduced an object manager class for each class that would be manipulated in our new system. The manager implemented methods to create, retrieve and store objects while ensuring the integrity between the two systems. The manager methods also checked and ensured that a particular object was only retrieved once. A manager class must only be instantiated once and we ensured this by using the Singleton pattern [5].
- Classes that could not be updated in our system were implemented as data banks. The data banks provided methods for looking up information in the legacy system. Persons and companies were not administrated by our system and we thus implemented a data bank for persons and a data bank for companies.

The mapping between business classes and encapsulation classes leads to high coupling between our new functionality and the legacy system. We solved this problem by using a Factory Method pattern [5]. Our business classes describe the general interface used by the presentation layer and they only implement general methods. Methods that depend on encapsulation classes are implemented in derived implementation classes. The principle is illustrated in the figure below:



This technique proved to be very useful. In principle, we are able to transform our system to use another equivalent legacy system.

5. Implementation

During the implementation, we experienced that it was comparatively easy to write the Java/CORBA code used to implement our design in spite of the fact that our knowledge of these technologies was limited.

Usually you do not have to think of a Java object being a CORBA object or not. But when parsing objects between CORBA objects you must be aware that only CORBA defined types (either standard types or IDL defined types) can be used as parameters to CORBA objects. A plain native Java object cannot be used. A JDBC database connection is a native Java defined class, and as such, it cannot be used as a parameter to a CORBA object. This caused a problem because we wanted several CORBA objects to use the same database connection, and thus needed to transfer the connection as a parameter.

Another issue to take into account, is that CORBA objects may reside in different processes. This is important to notice when objects contain non-data attributes - like an open socket or file descriptor. So even if we could make an IDL description of the database connection, it could not be passed across a process boundary, because it contains a connection to the database.

The solution was to introduce a single class by collecting all methods requiring a database connection for querying and manipulating the database. By designing systems with this kind of characteristics, we recommend to consider the connections to external resources.

5.1 Experiences with Oracle Application Server

Eventhough, we knew little about CORBA it was very easy to add CORBA functionality to our Java classes. This was due to the fact that Oracle has hidden the CORBA layer within the Application Server. When a Java class should be implemented as a CORBA class we needed to write the interface specification for this class. The interface is a Java interface that extends a special JCORemote interface that Oracle has defined. Then all we had to do was to deploy our Java class and interface specification to the server in order to make it a fully functional CORBA class.

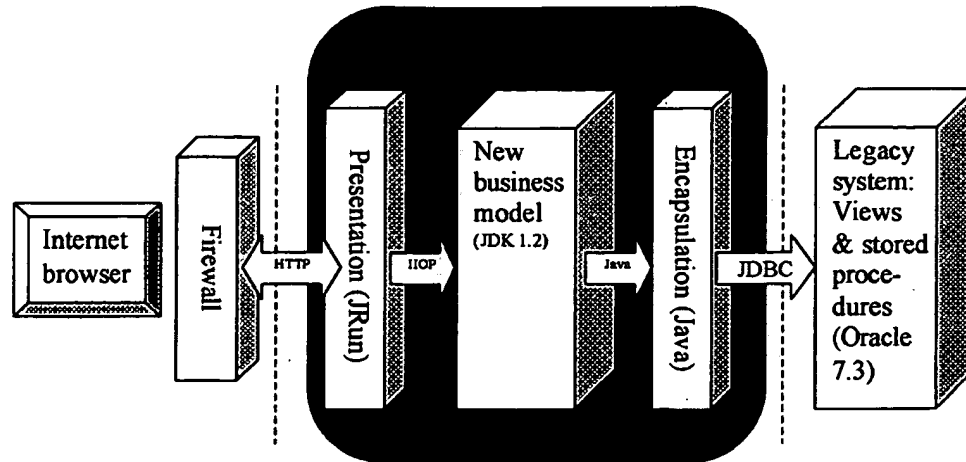
However the platform caused us several problems.

- In order to implement the Singleton pattern [5] for the manager classes we needed methods for storing and retrieving object references. A name service will usually be used to implement the methods but OAS does not support CORBA Name Service properly to be used for this. Other implementation techniques also failed and we were not able to find a solution that worked in OAS though it was crucial for the implementation.
- OAS has its own approach in support for CORBA functionality. This means you cannot use a lot of the existing literature and examples on Java and CORBA like in [6]. You have to use the CORBA documentation from Oracle and we found several situations where it wasn't sufficient.
- During the deployment process of Java classes to CORBA classes on the server, we experienced that OAS failed to deploy the classes when the number of classes exceeded a certain number. The reason seems to be a memory error, and we had to split our Java application into several applications on the Application Server in order to solve the problem.
- The automatic IDL generation in OAS reports very little on errors that might occur during compilation. This makes it very difficult to detect compilation fails.
- It takes a comparatively long time to deploy even a "small" application in OAS, which makes the edit-compile-test cycle a very heavy process for the developers.
- Finally we experienced a bootstrap problem, because some CORBA helper classes are generated on the time of deployment. The problem is that you need the helper classes when you compile your Java code i.e. before deployment.

The problem in implementing the manager classes as Singleton classes prevented us from getting the system to run with OAS. Though a solution probably exists, we were disappointed not to be able to find it within the project period. Thus we were forced to switch the implementation platform in order to get the system running.

5.2 Switching the ORB

Because of the problems experienced with OAS we decided to switch the implementation platform. The new platform was based on public available tools that we could download from the Internet and is illustrated below:



We decided to use the built-in ORB from JDK1.2 [7] to implement the business classes. We also decided to implement the presentation layer as plain Java servlets, which are Java classes that dynamically generate HTML pages for a web server. We selected JRun 2.3 from Live Software Inc. [8] to handle the servlets. JRun is a Java servlet (and JavaServer Pages) engine. It also includes a built-in Web Server that we used to test our application.

Moving the developed code from OAS to the new implementation platform went smoothly though it required code changes, primarily eliminating OAS specific details. The changes made the code more conformant to traditional CORBA/Java development and involved the following:

- We had to write all the IDL ourselves and generate it to obtain the Java classes. We used a trick to catch the temporary IDL files while OAS was deploying the application. These IDL files were then edited to exclude OAS details, and then generated for Java. In other projects the IDL design is an important activity and our OAS trick did not give the most positive solution, but the solution worked!
- The creation of CORBA objects was different. The new code was more simple and should only make a new Java object instance, and connect it to the ORB. We were pleased to find that these changes were isolated to the factories because we used the Factory Method pattern (see section 4.3).
- The Singleton pattern was implemented using the CORBA Naming Service.
- The presentation layer was completely rewritten to Java servlets. This may sound like a major task but the calls to methods in business classes and the generated HTML code were the same and could thus be plugged into the servlet code.
- Minor changes were needed like extending the implementation classes from the generated base classes.

In general these changes were caused by our initial choice of OAS. We believe fewer changes are required to switch between other ORB implementations that conform better to the CORBA standard.

One benefit from an application server like OAS is that it includes facilities to set up multiple applications and configure how they shall run in different processes. It is designed to handle large applications and optimise performance for the created objects. No equivalent tool is delivered along with JDK and we had to implement this ourselves. A simple server program was developed so that all objects were created within this server process.

We found that debugging was a lot easier on the new platform because we could see the output from the server process.

6. Conclusion of experiences

We have introduced a good method to extend legacy systems with new functionality. The use of object-orientation made it relatively easy to build an encapsulation layer on top of the legacy system. We believe the principles can be used in general.

Eventhough, we had little knowledge of this technology we benefited of being mentored by an external resource. We also had success with the use of patterns.

CORBA and Java give a fine platform to implement the architecture we decided to use. CORBA is almost hidden to the client and partly for the server application too. However, in the implementation design you must be careful only to pass CORBA defined data types in calls to CORBA objects. A consequence seems to be that the design must introduce one monolithic class with methods for all the functions requiring a database connection because a JDBC connection is a native Java defined class. We believe our observation is general for similar applications based on Java and CORBA.

We were disappointed with our experiences with the CORBA support in the current version of Oracle Application Server. The Singleton problem actually prevented us from implementing our design on the platform. A key problem is that the product tries to hide CORBA too much for the developer. This makes it difficult to use traditional CORBA techniques and existing literature. We strongly believe that it is better to use tools that really conform to the standard.

We were pleased to experience that switching the ORB platform required very few changes in the code. Having spent more than 600 hours trying to get the system running on OAS it only took two persons one day to switch to the ORB in JDK1.2. It really proves the benefits from using standards like CORBA!

Performance considerations had low focus in our project. The use of the new system will not be very intensive and thus performance is not very critical. We expect the implementation will perform adequately but in a mission-critical system you will probably have to spend extra time checking for unnecessary communication overhead between different objects.

7. References

- [1] Centre for Object Technology, www.cit.dk/COT.
- [2] RAMBØLL Information Technology, www.ramboll-it.com
- [3] Allan. R. Lassen. *WorkSAFE/IKIS empiri*. COT/4-09-V1.0. (Internal working paper in Danish).
- [4] Johnny Olsson, Lisbeth Bergholt, Aino Cornils. *Mønstre - en indføring i analyse-, design- og arkitekturmønstre*. COT/4-07-V2.1. (In Danish).
- [5] E. Gamma, R. Helm, R. Johnson & J. Vlissides. *Design patterns: Elements of Reusable Object-Oriented Software*, Addison-Wesley, Reading, MA, 1994.
- [6] R. Orfali, D. Harkey. *Client/Server Programming with JAVA and CORBA*. 2nd edition. Wiley, N.Y., 1998.
- [7] Java Development Kit Software, <http://java.sun.com/products>
- [8] Live Software Inc., www.livesoftware.com

Transparent Caching of Web Services for Mobile Devices

Kamal Elbashir
Multi-Agent Distributed Mobile and
Ubiquitous Computing Laboratory
Computer Science Department
University of Saskatchewan
(306) 966-4744
kae501@cs.usask.ca

ABSTRACT

A Web service is a collection of functions packaged, published and consumed using standard Internet protocols. This paper presents a generally applicable architecture for transparent Web Service caching, with a focus on Mobile Devices. This approach defines a set of required Semantics embodied in a document called the 'Service Semantics Description' (SSD). The SSD is used by the Client-side cache and (optionally) by the intermediate Caching Proxy. A tool facilitating definition of semantic tags is developed as an IDE extension. Preliminary results are presented outlining the feasibility, transparency, general applicability, and initial measurements of CPU and Memory overheads.

1. INTRODUCTION

Consuming a Web Service is accomplished by sending service requests (method calls), the service responds with a method return. Requests and responses are couriered as SOAP (Figure 1) messages (Simple Object Access Protocol). Accompanying a Web Service is the service's WSDL (Web Service Description Language), specifying syntax metadata. An optional layer, the Universal Description and Discovery Interface (UDDI) offers a directory service for discovery of web services [3, 4, 7, 12].

Communication with a Web Service is established using standard Internet protocols (e.g. HTTP). HTTP is the protocol underlying the World Wide Web. Adoption of Web Services gains remarkably from the utilization of well established protocols of the heterogeneous network of the WWW [4, 12].

A Web Service client must obtain the service's WSDL document. The WSDL document contains syntactic information about the service (Namespaces, Method signatures, Data types and a URI). A client consumes a web service by initiating the exchange of SOAP messages (see Figure 1-A). The messages must follow the inscriptions detailed in the WSDL document (URI, Signatures, and Data Type information) [4, 12].

For a mobile device, consuming a web service is an attractive interoperable approach to access remote business logic. Mobile devices are constrained by Battery, CPU, Memory and Weak Connectivity. Weak Connectivity results from intermittent communication due to low bandwidth, high latency or expensive networks. Furthermore, Mobile devices are susceptible to both voluntary and involuntary disconnections (user initiated disconnections vs. disconnections due to a change in the availability of a resource, e.g. Battery life). For the mobile user, constraining productivity to times of full connectivity hinders a successful and usable deployment of remote business logic [1, 7].

This paper will refer to a mobile device consuming a Web Service (a Web Service Client) as a Mobile Host (MH). A Cache is a temporal memory coordinated by a Caching Policy, cache records are memorized instances of data (Requests/Responses). A Proxy is an entity acting as an intermediate connectivity tunnel, a proxy maybe caching (Caching Proxy) [6]. A caching proxy plays an intelligent role while tunnelling requests to their respective endpoints. Cacheability of a request (method call/return cacheability) is the property stating that the request maybe cached while maintaining application logic (no negative side effects on the application logic) [3, 8].

2. PROBLEM DEFINITION

2.1 Introduction

Operation of a mobile service client is restricted to times of connectivity. The service Provider may suffer network or system downtimes. Furthermore, the mobile device is constrained by Weak network connectivity. When the mobile client is disconnected (null connectivity) the service is inaccessible, and the mobile application is unusable. Null connectivity occurs when the device is out of network range, or when the device is voluntarily or involuntarily disconnected (e.g. user-initiated power-off vs. a depleted battery state). Additionally, network infrastructures supporting mobility are bandwidth-limited. Clients consuming rich services suffer degradation of response times due to latencies incurred when sending large upstream requests (service arguments) and receiving of large responses (return values).

This paper will explore the issues present when a MH loses network connectivity (null connectivity) and the necessary recovery logic upon reconnection (reintegration phase). The user experience of a mobile device is reflected by the user's inability to continue working while the MH is disconnected. An improvement of the user experience can be achieved by enabling the user to continue to work while in disconnected mode. Increased application response times, due to caching of frequent requests and/or Prefetching of anticipated future requests is a desirable improvement.

The mobile application is assumed to be resilient to stale resource access (invalid, out of age resources). This implies that the application logic is not broken when a request is answered from the cache. This is a necessary assumption since the consistency of a cached resource cannot be guaranteed while operating in null connectivity.

2.1.1 Web Service Caching

A Web Service, described by the WSDL document can be viewed as a Remote Object, accessible by SOAP messaging over HTTP. Methods and properties exposed by the remote object are accessed using SOAP messages adhering to the calling convention prescribed by the WSDL. Every SOAP request/response corresponds to a single method invocation or a single property set/get operation [3, 4, 7, 12].

A naive caching architecture stores tuples of request/response pairs. When a new request is made; and a matching request is in the cache, then the corresponding response is returned (cache hit). If a similar request is not in the cache (a cache-miss); and the MH is operating in disconnected mode; then an exception is raised. All attempted requests, while in disconnected mode, are stored in a 'Pending' FIFO queue for execution when connectivity is restored (Reintegration Phase). Upon reconnection, the reintegration phase commences by issuing pending requests, an attempt of synchronizing the state of the disconnected cache with the state of the remote object.

The relationship between method calls and the state of the service is crucial. Service methods fall into one of three types: Read-methods (state-reading) and Write-methods (state-altering), or State-independent methods. Properties offered by the service are inherently state-reading and state-altering [3, 8]. The following discussion is only concerned with service methods; the outcome can be generalized to service Properties.

The classification of a method as state-reading, state-altering, or state-independent is best known by the publisher of the Web Service, since services implementations are exposed as black-boxes. It is impractical to assume that all methods are of one type or another (in reference to their state dependency). There is no standard for specifying Semantic information about Web Service methods. The WSDL document is limited to describing only the Syntactic information relating to service consumption [3, 8, 12].

A Web Service caching architecture, built specifically for Mobile devices must consider the limited processing and space constraints available to the mobile device (MH). These constraints limit the allowable Cache-size, and the processing requirements of cache management and operation [1, 3]. The architecture should preserve the logic consistency of the mobile application, a mobile application functioning with and without a cache should experience no side effects resulting in incorrect operation. Out of age cache records should be invalidated. Cross-MH cache consistency is a requirement for mission critical mobile applications (e.g. shop floor applications) [1, 3, 8].

Caching in the area of the World Wide Web is focussed on Read-caching [5, 6]. Read/Write caching has been well explored in the areas of File systems, Databases and Distributed Object Systems [2]. Many issues faced by a mobile Web Service client are comparable to aspects of caching in the aforementioned paradigms. This paper explores previous research in section 3.

Approaches common to Web caching are severely limited when applied to caching of Web Services. Static web pages need only Read-caching. Only when a sophisticated web caching system is utilized (e.g. Active Caching, for dynamic web content) then a form of Write-caching is present [5, 6]. A modified web caching proxy may treat the Web Service as a dynamic page and cache the SOAP messages that are couriered in the body of HTTP requests. Such architecture is very limiting and its assumptions are unrealistic. The modified caching proxy must compare bodies of

SOAP messages when testing for a cache-hit, the message body may include metadata that is not necessarily part of the method call (e.g. RequestID), resulting in false cache-misses [3]. Another problem with such an implementation is the proxy's inability to replay state-altering requests upon restoration of service availability. Finally and most importantly, the meaning and structure of a web page is fundamentally different from that of a web service, the required semantic information enabling caching is hardly translatable into the domain of web caching.

2.1.2 Mobility and User Experience

A cache for mobile devices must support transparent switching between connected mode (remote-execution) and null connectivity mode (cache-based execution).

Furthermore, service methods requiring large arguments incur delays due to network transfer latencies. Similar latencies are incurred when methods return large dataset are executed. Repeated duplicate requests should only be returned from cache and bandwidth utilization should be minimized when possible [7, 8].

2.1.3 Caching-Architecture Deployment and Existing Web Services

A caching architecture may require the deployment of specialized components on the server-side, a difficult requirement in real-world scenarios, service Providers are generally reluctant to alter existing implementations. An approach to transparency is achievable by caching independently of the service implementation, utilizing a Caching Proxy [7, 8]. Caching schemes employing an intermediate proxy are extensively used for caching in the areas of the WWW and Distributed Object Systems. An intermediate proxy implementation appears to be the service client when viewed by the service provider. When viewed from the client's perspective, the proxy appears as the original Web Service.

2.1.4 Cache Location

A client-side cache (see figure 1-B) permits the client application to recover from service requests when the service is unavailable (network outage or service downtime). If the client is caching independently of the service, unnecessary executions on the server if the client issues an expensive request and network connectivity is lost before a service response is received. Furthermore, notifications of invalidated client-cache records are harder to implement since the server is unaware of the client cache. Finally, multiple caching clients are not centrally coordinated, the clients maybe within range to form an Ad-hoc network and exchange cache-hits while the service is unavailable.

A second cache (see figure 1-C), residing at an intermediate layer between the mobile device and the web service is a promising approach. The intermediate cache (caching proxy) must not be susceptible to network disconnections (see Figure 1). Such an organization permits proxy-led coordination of client caches. The service is now shielded from processing duplicate requests made by multiple mobile devices. A proprietary protocol, optimizing the link between the intermediary proxy and the mobile device is now possible (e.g. offering compressed streams). Invalidation reports are more readily deliverable to mobile clients, as the intermediate caching proxy becomes the centralized cache coordination layer.

2.2 Key Questions

From the previous discussion, several questions arise:

1. What metadata is necessary to enable caching of Web Services? How can I enable a developer to specify the necessary metadata with minimal effort?
2. What should be cached?
3. How can a cache that is independent of service implementations be implemented?
4. What consistency guarantees are attainable?
5. What prompts cache invalidation? What happens in the reintegration phase?
6. How does existing research help in designing a practical caching architecture for Web Services?

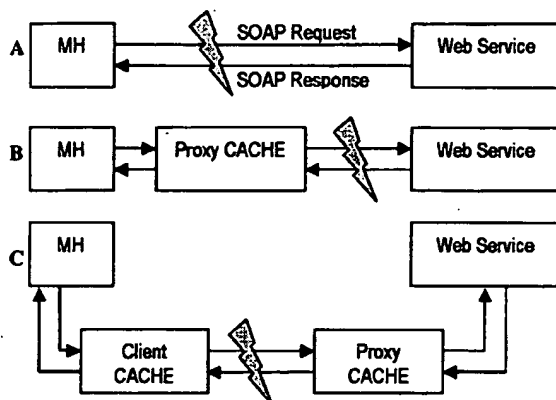


Figure 1: MH - Web Service Connectivity without a Cache

3. RELATED WORK

3.1 Web Caching: A Survey of Web Caching Schemes for the Internet

Wang's paper [5] is a survey of caching architectures on the Web. Proxy servers are recognized as an effective solution for bandwidth optimization, availability, and scalability. Web Proxy servers sit between the web client and the web server, mimicking the behavior of the real server. Caching may occur on the server, the proxy or the client. A set of significant improvements are recognized as following [5]:

1. Web caching optimizes bandwidth usage by reducing network traffic as more and more resources are found in the cache.
2. Web caching reduces access latency due to the fact that documents may be found in the local cache (on the client) or on nearby proxy servers, reducing required network traffic when fetching documents. Because of the traffic reduction, more bandwidth is available for a cache-miss to be retrieved quickly.
3. The web server's performance is improved as more requests do not reach the server and documents are retrieved from the client or proxy caches.

4. Improved robustness, document availability is significantly improved due to replicas existing in local and remote caches. If the server goes offline, cached documents are still available to interested clients.

5. Proxy servers act as intermediary traffic deltas, data collection and statistical analysis of access patterns is more easily doable.

6. Load balancing is in effect, as proxy servers reduce the server's utilization by forwarding requests to the least utilized server.

Wang recognizes disadvantages of web caching, most importantly is cache consistency maintenance, as stale records may be served to clients while the server is offline. Increased request overhead, due to processing by intermediary proxy (or proxies). A proxy is also recognized as a point of failure, the proxy's availability becomes more critical than the web server's availability. This is because one proxy server may act as a caching agent (delta) for multiple web servers [5].

Wang outlines ten desirable properties of a web caching architecture, namely: Fast Access, Robustness, Transparency, Scalability, Efficiency, Adaptivity, Stability, Load Balancing, Ability to deal with Heterogeneity and Simplicity. A number of caching architectures are analyzed including Hierarchical, Distributed and Hybrid. Hierarchical caching producing the shortest connection latencies when compared to distributed caching. On the other hand, distributed caching is found to have the shortest retrieval latency but with greater bandwidth utilization [5].

Two common approaches to cache routing and optimization are recognized, by growing a caching distribution tree formed by nodes of intermediary proxy servers. Cache resolution is performed by a routing table or a hash function [5].

Prefetching [9] is needed in order to maximize the cache-hit rate, this is done by anticipating future requests and prefetching the corresponding documents pre-demand. Prefetching may be executed between the client and the web server, between the client and the proxy agent, or between the proxy agent and the web server. Caching between the client and the web server is recognized for increasing the cache hit-rate by 45% at the cost of doubling network traffic. Rate-controlled prefetching is recognized as a possible solution to the unacceptable increase of network traffic [5].

Prefetching between the proxy agent and the web server provides significant improvements over the previous approach offering a very good predictability of client access patterns at the proxy [5]. Furthermore, due to the decreased network traffic downstream to the client. Prefetching between clients and proxy agents may best support caching for mobility, as the client and the proxy agent cooperate to cache and prefetch documents without the server's intervention [5].

A number of cache placement and replacement strategies are identified and analyzed in regards to cache size/speed. A good cache placement/replacement strategy is very important when caching for mobility. Two types of cache coherencies are recognized, Strong and Weak. Strong cache coherency is maintained either by client validation or Server invalidation. The former results in higher cache-hit latencies, while the latter requires server cooperation. Server cooperation may be needed for broadcasting invalidation reports. Weak coherency is the case when cache records are invalidated by a timeout (TTL value). Piggyback invalidation requires the server cooperation, by

sending lists of invalidated items attached to responses of new requests. The requirement for cache coherency depends largely on the client's tolerance for stale resources and on the frequency of resource changes [5].

Proxy placement is very critical for optimal performance gains. The desired properties inherit many of the desired properties of a web caching system [5].

Caching of Dynamic web resources, a topic of interest when considering caching of web services is briefly touched upon. Read-caching is the focus of the majority of web caching strategies and studies [5]. The two widely used approaches to dynamic-resource caching are identified as Active Caching and Server Accelerators. The former requires computation applets attached to dynamic portions of a page to be sent to the proxy and the latter is done by exposing a set of caching APIs at the accelerator entity, the dynamic resources at the server must utilize the APIs in order to achieve caching of dynamic resources [5]. Both approaches violate at least two of the desired properties outlined at the beginning of the paper, namely the ability to deal with heterogeneity and simplicity.

3.2 Web Service Caching for Mobility

Terry et al. [3] discuss the need for caching of Web Services to support mobility. The motivation is to offer disconnected operation of web services. Transparent deployability and General applicability are two desired properties of a web service cache. The paper distinguishes attributes specific to caching of web services, when compared to caching in the areas of file systems, and databases as both offering standard interfaces with well known semantics (Read\Write and Query\Insert\Update\Delete), and Web caching is limited to read caching [3].

The authors use the .NET MyServices set of services for an experiment in caching for disconnected operation. .NET MyServices are services offering personal profile maintenance and a contacts directory. The exposed operations are Query, Insert, Update and Delete operations [3].

The proposed architecture utilizes an intermediate SOAP proxy agent, caching SOAP requests/responses made by service clients and serving cached responses when the client is in disconnected mode. Cached requests are queued up for replay (playback) at the reintegration phase [3].

Two critical issues surrounding caching of Web Services are identified, namely Cacheability/Playback, and cache Consistency maintenance. For an effective web service cache, the authors identify the requirement that all service operations must be designable as Update or Query operations (Read\Write semantics). The lack of semantic information in the service description (WSDL) creates a challenge for caching consumers of boxed web services. Query operations are deemed cacheable if they do not alter the server state (e.g. server logs). Update operations are operations that are state altering on the server [3].

Maintaining strong cache consistency on a mobile client is deemed unachievable by the inherent property of weak connectivity. Consistency is also explored when execution of an operation invalidates a stored response of another. A proposed solution is to invalidate the old record, or apply a modification transformation for the in-cache record. The authors declare that "for preexisting Web services, understanding the correct consistency requirements is an extremely challenging issue" [3].

The effect of a web service cache on the User experience is identified as a crucial criterion when evaluating an effective web service cache. Ideally, the user should not be aware of disconnections but this is identified as a difficult goal. An additionally challenge is the ratio of consistency guarantees offered by the cache and the quality of the user experience. Altering the client to be cache-aware, and to display hints to the user regarding the active consistency guarantees may have a positive effect on the user experience [3]. The later proposition does not satisfy the property of transparent deployability outlined at the beginning of the paper.

Terry et al. discuss the effect of differing structural formatting of SOAP messages on a web service cache. This is identified as a possible problem when comparing requests for similarity. WSDL is identified as providing enough information for an intermediate proxy to fabricate a fake response to a service request. Although the lack of a specification mechanism for Default values may limit the range of possible fabricated responses [3].

Prefetching (hoarding) is identified as a mechanism to maximize cache-hit rates. An implementation would employ an algorithm for anticipating requests for prefetching. The lack of semantic information about the service and the lack of a standard mechanism for users to specify a set of requests for hoarding complicate a cache implementation supporting prefetching [3].

Finally, maintaining application Security is outlined as an important consideration. Though is complicated by lack of standards regarding authorization of access to a web service operation. An intermediary Proxy, caching for multiple clients, may open a set of privacy/security holes, this is relevant when cached responses differ by the authenticated user [3].

3.3 Caching of Objects in Distributed Object Middleware (CORBA) for Mobility: Domint

Distributed object middlewares offer remote method execution, platform interoperability, and location-transparency of objects. The first two properties offer the closest resemblance of the Web Services paradigm.

Conan et al. [2] describes an architecture offering disconnected operation of a CORBA environment for mobile clients. Domint, uses portable interceptors (PI), a CORBA mechanism for peaking into and altering of the communication between a client and an ORB (Object Request Broker). Domint offers continued operation in partial and null-connection modes with minimal or no overhead when operating in connected mode.

The transparency of utilizing CORBA's portable interceptors enables connection awareness to be shifted away from both the client and server objects and into the Domint middleware extension. Domint works by intercepting requests made to the CORBA ORB and transparently rerouting requests to a local disconnected object [2].

Several performance protective measures are employed. In order "not to punish strongly-connected clients", while strongly connected, client's requests go directly to the remote object [2]. Also a hysteresis mechanism is proposed for handling variations in connection availability. An interface to the hysteresis mechanism maybe consumed by the client application in order to alert the user to changes in the connectivity-mode, also offering the user the ability to voluntarily disconnect. Three connectivity modes are recognized, namely: disconnected, partially connected, and connected. Transparent switching between modes is activated

at the time of a client request. A set of inputs is required when deciding on an operation to execute, the inputs are: disconnection mode (voluntary or involuntary), the mode of the last request (to the same object), the operation name, and the networkobject current connection mode. A matrix is developed allowing correct state transfers in various modes [2].

In the connected mode, the requests are immediately sent to the remote object. In the partially connected mode, the operation is executed both locally and remotely, depending on the call semantics (presence of in, out, in/out parameters and if a return value is expected). In disconnected mode, operations are executed locally, and are logged depending on the semantic relationships with other operations. The log is vital at the reintegration phase and reconciliation may need to occur to maintain coherency between the disconnected object (the proxy) and the remote object. However, Domint assumes that no object is accessed by more than one disconnected client [2].

Preliminary performance evaluations, performed on a Windows CE device, show overhead of 14% to 1% when connected, 50% to 6% when partially connected, and from 20% to 6% when disconnected. The incurred computation cost is justified by the introduction of transparent connectivity, without modification to either the server or the client implementations [2].

3.4 Delayed Execution/Call Aggregation: Reducing Overhead of .NET Remoting

In the context of Web Services, .NET Remoting is the infrastructure implicitly in-use when .NET applications publish or consume Web Services. Remoting abstracts remote objects to behave as local objects. The Remoting infrastructure offers various extensibility options, the lowest level communication channel maybe replaced or customized, the messages to be exchanged maybe modified before or after formatting, and calls to remote objects maybe intercepted immediately after a consumer issues a request and before the request is propagated downwards in the remoting stacks. The later is the mechanism commonly used in distributed object middlewares. The client accesses the remote object via a local proxy, known as the Transparent Proxy in .NET Remoting. The transparent proxy is generated at runtime by the Real Proxy (also a client-side object). The real proxy is generated when remote objects are referenced, and its binary maybe replaced or modified without modification to the object consumer.

Clegg [11] discusses the overhead introduced when employing .NET Remoting for remote execution. Clegg describes and evaluates an architecture that transparently monitors and optimizes calls to remote objects. RROpt is modeled on the DESORMI framework (Delayed-Evaluation, Self-Optimizing Remote Method Invocation by Kelly, Field, Bennett, and Yeung). The implementation is a modification to remoting-relevant code in the .NET CLI, namely the Mono CLI. Such an implementation eliminates the need for server/client modifications [11].

PROpt works by checking at runtime for candidate delayed calls, specifically by looking for methods of objects inheriting from MarshalByRefObject. When a remote call is incurred, it is stored in a delayed-list and a dummy return is pushed to the stack. If a method attempts to use the return value then the delayed method is immediately executed. A set of delayed methods is executed by formulating a plan encapsulating their data dependencies and forwarding the plan to the server. PROpt assumes that all servers are PROpt-enabled (executing on top of Mono CLI with PROpt

extensions). Argument aggregation is also performed when a set of methods share an argument [11].

Another optimization employed by PROpt is Plan caching, sets of previously executed aggregated-calls (a plan) are remembered on the server, a client refers to them by ID, furthermore decreasing network traffic requirements [11].

The remoting infrastructure protects applications by containing them in "Application Domains" (light weight processes). Multiple client accessing the same remote object are not aware of each other, multiple application domains may be hosted within one process. Furthermore, multiple remote objects on a single remote server maybe accessed by a set of application domains in a process. In order to aggregate cross-object, PROpt implements its aggregation targets per server name [11].

The speedup possible by PROpt did not prove to be consistent. PROpt performed well when data dependencies between methods existed. Outbound Network traffic is significantly decreased due to call/parameter aggregation. PROpt optimizations failed to materialize when no data dependencies exist between method calls, this is the case when the network infrastructure is fast [11]. No applicability to mobile clients is considered.

4. A WEB SERVICE CACHE

4.1 General Considerations

An architecture supporting a predefined set of services and a special client implementation is an application-specific cache. An application-specific cache is optimized for the application logic, all caching decisions (such as placement, replacement, prefetching) target optimal consistency and performance of the application. Proprietary communication protocols (such as ones supporting compression, or multicast notification of cache invalidation) are expected, as the application permits [3, 8].

A General caching architecture, on the other hand, offers a cache to any Web service. Such architecture faces many challenges, most importantly is the decision of cacheability of a web service request. A Web service is treated as a black box, requiring the availability of cache-hints (metadata) supporting decisions such as cacheability, invalidation conditions, and default responses (return values) [3].

A caching proxy is necessary in a general cache, in order to support independent caching decisions (independent of the client and the server). On the other hand, an application-specific implementation maybe embedded in the web service and client implementations.

When the goal of a cache is to improve the availability of a web service, then a larger Cache-size and optimal placement and replacement strategies are a priority. The existence of stale-resources (invalid cache records) in the cache is permitted, hence to improve the service's availability. Knowledge of the MH connection-state is necessary, in order to seamlessly resume returning of cached responses when the MH enters null connectivity. Cached responses are returned only when the MH enters null connectivity, in order to achieve the best-possible cache consistency. Newer requests overwrite their older counterparts in the cache.

On the other hand, when the goal of caching is to improve the application's performance, then cached responses can be returned even when the MH is in full connectivity mode. Such an approach may result in substantial response-time improvements, especially

for expensive web service methods: methods requiring a relatively large set of arguments, methods requiring an expensive or lengthy computation, and methods returning a sizable data object. Web service requests maybe aggregated to improve bandwidth utilization while returning cached responses. The downside to such performance optimizations is decreased cache consistency in relation to the real web service. A workaround to further improve the consistency of cached responses maybe achieved by periodically prefetching, or periodically submitting invalidation-queries of expensive cached responses.

4.2 Cache Location

A server-side cache offloads the server from re-computation of frequent requests. Such a cache implementation is commonly a specialized architecture. Cache-consistency is best obtained when using this approach, since invalidation reports maybe requested or broadcasted to the known caching proxy (or proxies). Another improvement is the transparency of the cache for a MH consuming the web service. A slight improvement in service availability is present due to a protection from server downtimes, as the server-side cache continues operation while the server is down. On the other hand, a MH suffering local null connectivity is also disconnected from the server-side cache.

A client-side cache offers the service's availability to the MH while in null connectivity [10]. Cache-consistency is minimally maintained in this approach. Near-time cache invalidation is harder to achieve since the MH cache is independent of the service implementation [3, 8].

An intermediate caching proxy offers transparent service caching for a MH. A caching proxy is more capable of tracking the list of caching MHs as it act as an intermediate delta between multiple MHs and multiple service providers. A shared cache is in effect, as requests from multiple MHs are cached for other MHs. An intermediate cache is best equipped, independently of the service provider, to deliver invalidation reports to the tracked list of MHs. The service remains unavailable to a MH in null connectivity, as the intermediate cache becomes disconnected.

A client-side cache assisted by a caching proxy is best equipped when the goal is protect the client from service unavailability while maintaining best-possible cache consistency. Several performance improvements are now possible because the client-side cache and the intermediate caching proxy can agree on a proprietary communication protocol supporting better request aggregation and near-time broadcasts of cache invalidation reports.

4.3 Semantic Metadata

The lack of semantic metadata of a web service methods presents a challenge for caching independently of the service implementation [3, 8]. The metadata may accompany the service as an extension to the service's WSDL document, or maybe maintained by a third party maintaining a repository of metadata records targeting a growing set of a web services. An alternative is to allow the service consumer to specify an updatable set of meta tags, assisting caching decisions when determining cacheability and invalidation conditions. Client maintained metadata are specified per web service.

A web service method should be tagged if it is cacheable, and a default return value should be specified. The former aids the cacheability decision of the cache, while the latter offers a protection against cache-missed of a MH in null connectivity.

Additional tags may outline invalidation conditions based on time-values, age-thresholds. Method interdependencies will aid request-aggregation logic and can also provide further improvement when maintaining cache consistency by invalidating cached responses when state-modifying requests are executed.

4.4 Caching Policy

The request signature and argument values must be considered in hash function supporting cache placement of web service responses. Since multiple requests to the same method may differ on a single argument, while unique responses are always returned.

LRU, LFU and Size are competing replacement strategies when a decision relates to limiting the cache size, especially for a MH with space and computation constraints [7]. It is to be determined if any or a combination of the well studied replacement strategies are best suited for a web service cache supporting mobile devices.

A cached response maybe invalidated by its age or a timeout value. Furthermore, a cached resource maybe invalidated and evicted from the cache because an invalidating request was submitted.

Cache invalidation reports maybe broadcasted by the server or an intermediate proxy, or maybe piggybacked on the results of new requests. Furthermore, invalidation query maybe submitted on intervals or piggybacked on new requests. Limitations are present depending on the cache location and the number of entities supporting caching between the MH and the service provider [3, 8, 10].

Default return values are useful on a cold-start or when a cache-miss occurs while the MH is in null connectivity mode.

An alternative for recovery from a cache-miss when the original service is down while the MH is fully connected is by rerouting of requests to a service replica. This approach further protects the MH from service unavailability due to provider downtimes or peak hour unavailability.

The consistency of responses between the service replica and the original provider is an issue out of context for this research.

4.5 Prefetching/Hoarding

On a cold start, a MH may issue a set of predefined requests for caching. Alternatively, the service provider or an intermediate cache maybe store an up-to-date image of most frequently requests methods and push them to the client on a cold start.

Prefetching offers substantial improvements in response times to most frequently requested methods, at the expense of higher bandwidth utilization. The negative side effects of prefetching maybe overcame by request-aggregation and by adaptive prefetching logic with explicit awareness of network QoS [9].

4.6 Client Sessions

A MH utilizing a communication channel supporting session state can benefit from an intermediate cache retaining a MH-tailored list of frequent requests. Prefetch or hoarding requests can be initiated on the behalf of the MH.

For a frequently disconnecting MH, expensive requests made while in full connectivity maybe pushed upon reintegration. The minimum improvement is in effect when a cache-refresh cycle (or a cache-replacement function) is executed and an expensive request is not evicted because the owner MH connectivity is considered.

5. PROPOSED ARCHITECTURE

The design of the proposed architecture is modelled to support the following two scenarios and their consequences:

1. Null Connectivity: The MH enters null connectivity, on a new request the following conditions are evaluated:

A) Cache-hit, the cached response is returned. A state-altering request is queued into the Replay queue. State-reading requests are queued into a Delayed-fetch queue, a mechanism for improving cache consistency at the reintegration phase.

B) Cache-miss (Cold Start), a default return-value is returned. A state-altering request is queued in the Replay queue. A state-reading request is queued for delayed fetch.

2. Full Connectivity: The MH enters full connectivity, the following conditions are evaluated:

Non-empty Cache:

A) Cache Miss:

New requests go directly to the Web Service, and when a response is received; a request/response tuple is inserted into the cache.

B) Cache Hit: State-altering requests are sent directly to the service, a request/response tuple is cached. A response to a state-reading request is returned from cache (if the response is valid or if the request is long-living), the state-reading request is queued for delayed-fetch, a mechanism for improving cache consistency in the long-run. A long-living request is a request with a long cache TTL or the time (t) of the cached response is less than the invalidation time (On Time) of the method.

Empty Cache (Cold Start):

A) On start, if an intermediate proxy exist, request a cache Image, Done.

B) On start, if a Prefetch-list is known, queue all items from the list into the Delayed-fetch queue, Done.

C) On a new Request and a Cache Miss: Return the default return-value associated with the request and queue state-altering requests into the Replay queue. State-reading requests are queued for delayed-fetch.

For effective caching of the Web Service this approach uses a document encapsulating the Semantics of the Web Service. The encapsulated semantics are shareable, and are either defined by the service Consumer (at development time) or by the service Provider. This paper refers to the document encapsulating the service semantics as the Service Semantics Description Document (SSD Document), an XML document. The SSD also specifies Hints aiding various cache operations (Invalidation Conditions, Prefetch Lists, and addresses of a Replica and Intermediate Proxy).

A developer tool that integrates within the IDE of Microsoft Visual Studio.NET is provided to aid a mobile application developer to seamlessly specify an SSD. A similar tool is also provided for a Web Service developer in order to specify an SSD. The developer tool allows transparent incorporation of a cached Web Service while decoupling the programmer from the caching infrastructure.

The Service Semantics Description document specifies the following metadata regarding each service Method:

1) Cacheability: specifying if a method should be cached or not. A method is non-cacheable if a cached value will always lead to faulty application logic (e.g. a GetLastRequest method).

2) Replay: upon reconnection, this tag hints if a method should be replayed or not. To maintain application logic, a State-altering method should be tagged for 'Replay'. A method may not be tagged for replay because of one of two reason; either the method is state-reading or the method's state-altering behaviour is irrelevant upon reconnection.

3) Default Return Values: the value of this tag is a serialized-graph of a meaningful default return value for a method. This tag enables the MH to partially recover from a Cache-Miss while in full connectivity, or when a MH cold starts. It is expected that only cacheable (state-reading) methods will have default return values. State-altering methods can not have default return values as this may lead to illogical returns (e.g. a Bool CreateRecord() method, returning success or failure of record creation).

4) Invalidation Conditions: a cached response should be invalidated after a specified Age (in minutes), or after a certain time of day, or when an invalidating hint is received (or fetched), the latter is not implemented.

5) Method Interdependencies: for N methods, this is an N x N table specifying interdependencies between service methods. Effective Request-aggregation can only be implemented if method interdependencies are known, the current prototype does not implement request-aggregation. A cached response is invalidated if a state-altering request, that is also a dependency of the cached response, is submitted. A table lookup is used to invalidate a cached response.

6) Prefetch Parameters: a list of method names and proper arguments is specified to enable prefetching on a cold start or at periodic intervals.

7) Intermediate Cache Proxy: a URI specifying the network path to an intermediate Caching Proxy, the caching proxy interface appears as a replica of the original Web Service.

8) Service Replica: a URI specifying the network path of service Replica. A Replica is utilized when a service downtime is detected.

The implementation of the Cache is built as a Proxy of the Web Service, the Proxy exposes an interface identical to the real service, decoupling the service consumer and service provider from the cache. The Proxy implementation is essentially a disconnected Object providing transparent access to the real service while continuously monitoring service availability and network connectivity of the MH. This object is referred to as the Caching Web Service Object (CWSO).

The CWSO implements a Hashtable for storing tuples of request/response pairs. A connectivity monitor detects service availability by requesting the service's WSDL at predefined intervals (5minutes). The connectivity monitor has OS hooks to detect network connectivity at the MH, entering the CWSO into one of two states; Full Connectivity and Null Connectivity. A Replay FIFO queue and a Delayed-fetch FIFO queue are used for

queuing state-altering requests (the former) and queuing of prefetch-requests (or delayed-fetch requests).

The SSD document is stored as an XML file accompanying the Real Proxy assembly. The CWSO provides an interface for consumers wishing to alter the service semantics or the default caching policy at runtime.

The Cache hashtable contains wrapped IMessage objects (CacheEntry). An IMessage object specifies the method's signature, argument list, call context, and method's response. The CacheEntry object includes the request time and time of the last cache-hit, along with the object's size in bytes.

The Cache Manager executes at state transitions (Full Connectivity and Null Connectivity) and appropriate action is taken. The Replay queue is processed before processing of the Delayed-fetch queue, to allow state changes to occur before processing of new state-reading operations. The processing of the Replay queue is started at a random interval between 1-5 minutes after achieving full connectivity. On a Cold-start, the replay-queue is empty and processing commences with the delayed-fetch queue instead.

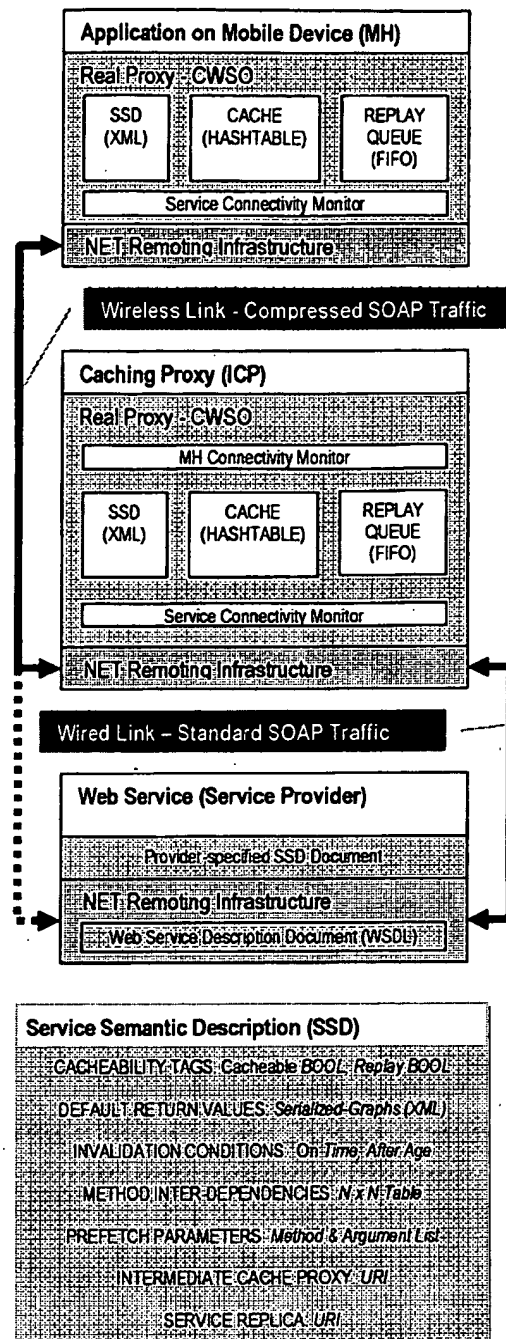
The Intermediate Caching Proxy (ICP) hosts an exact copy of the MH's CWSO, plus an additional connectivity monitor targeting MHs. Connectivity session management between the ICP CWSO and the MH's CWSO is planned as future work.

The link between the MH and the ICP is a wireless link susceptible to disconnection and low QoS (factors of weak connectivity), SOAP communication is tunnelled over a 'customizable' HTTP channel (e.g. Compressed). The link between the ICP and the real Web Service is assumed to be a wired link, offering higher bandwidth and strong connectivity, the communication is couriered by standard HTTP.

When the MH's CWSO is accessing either the real Web Service or the ICP, the logic is consistent. A replica Web Service appears to be the real Web Service to both the CWSO and the ICP. The MH or the ICP both appear as simple clients to either the Web Service or the Replica.

If a Service Connectivity Manager detects a service downtime (failure to return a WSDL document), requests are transparently routed to a Replica (if exist) or the Cache. If the connectivity manager detects null connectivity at the MH, then the ICP, the real Web Service and the Replica are all disconnected, and all requests are routed to the local Cache.

Cached responses are invalidated by Age or Time (from the SSD), the detection of an invalid CacheEntry happens when a cache-hit is suspected or when the CWSO executes the cache's SizeManager (every 20mins). CacheEntry-objects maybe evicted from the cache if the cache-size exceeds a threshold (predefined as 10mb), the eviction strategy maybe LRU or SIZE.



6. PRELIMINARY EVALUATION

6.1 Experimentation Plan

Two experiments have been performed, the first experiment measures the Overhead introduced by CWSO on the MH. The second experiment captures the State Transitions when controlling factors are toggled.

The scenarios are controlled by the following factors:

- Network Connectivity: Connected and Not Connected
- Service: Available and Unavailable
- Request: State-reading R, State-altering W
- Cache Test: Hit or Miss

The service provider in the Experiment 1 is on the same host as the emulated MH. This decision was made to eliminate network latencies from the experimental results. All performance data in this experiment is collected at the local MH.

The service provider in the Experiment 2 is a remote host. The collected performance data is local to the MH.

A mobile network was not utilized in these experiments, this should not skew the result sets since this implementation does not introduce additional communication.

The link between the ICP and MH's CWSO is a standard HTTP stream, SOAP Compression is not yet implemented.

Cooperative Caching and Prefetching has not been tested.

Network, Service, Replica and ICP availability is simulated by object parameters.

The active replacement policy is LRU.

6.2 Test Suit

6.2.1 Hypothetical Web Service (HWS)

The service exposes 4 methods, R denotes a State-reading method, W denotes a State-altering (write) method:

OUT R_1() consistently returns a constant value.

OUT R_2(in) is a function of in.

OUT W_1(in) is state-altering returning success/failure.

VOID W_2(in) is state-altering without return.

The 'in' argument to method R_2 is an Integer, W_1 and W_2 'in' arguments are of type String, the String arguments vary randomly in size between 100bytes and 1kb.

The return value of R_1 is a String, R_2 is an Integer and W_1 is Boolean.

The associated SSD is available in [13].

The goals of this experiment is to test processing overhead (CPU) and cache-size overhead (Memory) at the MH's CWSO when 1000 requests are sequentially executed.

To calculate CPU and Memory overheads, the experiment is run twice, for each method. The first run is performed without the CWSO, the second run is with the CWSO. CWSO initialization times are accounted for.

6.2.2 I-Help Web Services

I-Help is a real-world public discussion forum system, utilized mainly by Computer Science students at our department. Actual User Traces were not collected for this experiment, instead the

goal of this experiment is to verify the architecture's general applicability to existing Web Services and secondly to verify the system's State Transitions when controlling factors are toggled.

I-Help Web Services exposes two operations of interest, a Query operation and a Post operation, the associated SSD document is available in [13].

6.3 Experimental Conditions

Mobile Application: Proof of Concept Client

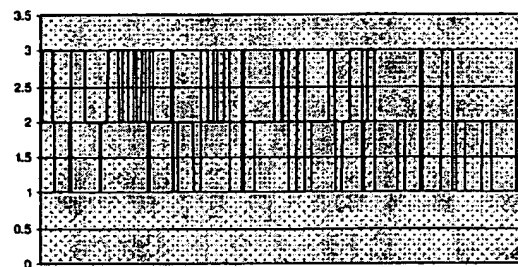
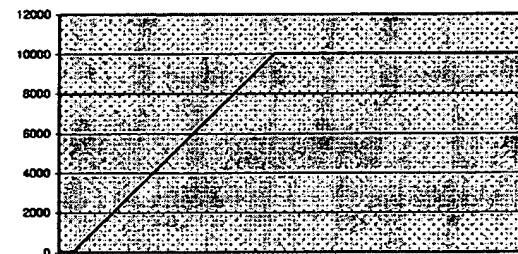
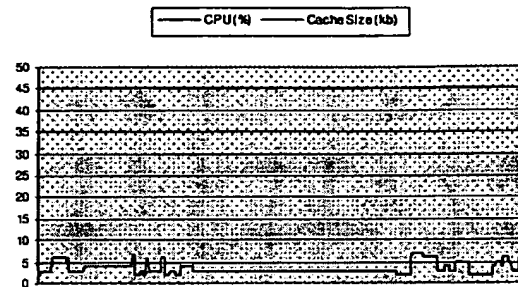
MH: Emulator Windows CE.NET 4.2.

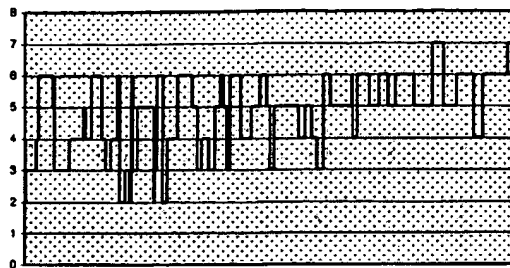
Experiment 1 Service Provider: .NET Assembly

Experiment 2 Service Provider, 3rd-party implementation: Axis, Java Web Services

6.4 Preliminary Results

6.4.1 Experiment 1: Overhead





CPU and Cache-Size, Experiment 1, Method: W_2

6.4.2 Experiment 2: State Transitions

Network Connectivity	WS Availability	Cache Hit	Operation	Execute at Service	Cache	Default	Delay Fetch	Replay
Y	Y	Y	R		1		1	
Y	Y	Y	W	1				
Y	Y	N	R	1				
Y	N	N	W			1		1
Y	N	Y	R		1		1	
Y	N	Y	W		1			1
N	Y	N	R			1	1	
N	Y	N	W			1		1
N	Y	Y	R		1		1	
N	N	Y	W		1			1
N	N	N	R			1	1	
N	N	N	W			1		1

6.5 Summary of Results

The results show that the CPU overhead at the MH, expended in the Real Proxy object by the components: Service Connectivity Monitor, Cache-Hit Test, Cache-Size manager and the Queues, did not rise above 10%, as a preliminary result this is acceptable.

The SSD's description of W_1 as non-cacheable successfully resulted in a Cache-size of 0bytes. W_2, marked with a VOID return resulted in a similar outcome. R_1 returning a constant occupied 45kb in the Cache, a value indicative of the size of initial CacheEntry object, Hashtable initialization along with a set of data owned by the .NET Framework's memory management. On R_2 execution, cache-size grew rapidly as random 'in' arguments were sent with every new request, the cache-size capped at 10mb, the predefined maximum allowable cache-size, future requests replaced in-cache entries by the LRU strategy.

The CWSO State changes match expectations. The detected States match the architecture's logical design. This experiment utilized a

real-world Web Service, demonstrating general applicability of the architecture.

7. FUTURE WORK

Embed the Service Semantics Description within the service's WSDL, this maybe done by utilizing WSDL Extensibility via attributes and element extensions. Merging the syntactic description (WSDL) with the semantic description (SSD) is very valuable, revoking the need for a separate SSD document and enabling smoother integration and richer discovery of Web Services.

8. CONCLUSION

This paper presented a generally applicable, connectivity-aware, and a transparent approach to caching of Web Services. A set of semantic tags have been identified as a prerequisite for effective web service caching. The Service Semantic Description document was developed, along with a tool enabling SSD-specification from within a widely used developer IDE. Preliminary evaluations of the architecture demonstrated general applicability, transparent operation and low resource overhead.

9. REFERENCES

- [1] D. Barbar, T. Imielinski. Sleepers and Workaholics: Caching Strategies in Mobile Environments (1994).
- [2] Denis Conan, Sophie Chabridon, Olivier Villin, and Guy Bernard. Domint: A Platform for Weak Connectivity and Disconnected CORBA Objects on Hand-Held Devices (May 2003).
- [3] Douglas B. Terry and Venugopalan Ramasubramanian. Caching XML Web Services (May 2003).
- [4] Jen-Yao Chung, Kwei-Jay Lin, Richard G. Mathieu. Web Services Computing: Advancing Software Interoperability (2003).
- [5] Jia Wang. A Survey of Web Caching Schemes for the Internet (1999).
- [6] M. Baentsch, L. Baum, G. Molter, S. Rothkugel, and P. Sturm. Enhancing the web infrastructure - from caching to replication (April 1997).
- [7] M. Tian, T. Voigt, T. Naumowicz, H. Ritter, J. Schiller. Performance Considerations for Mobile Web Services (2003).
- [8] Matt Powell. XML Web Service Caching Strategies (April 2002).
- [9] Pei Cao, Edward W. Felten, Anna R. Karlin, and Kai Li. A study of integrated prefetching and caching strategies (June 1995).
- [10] Roy Friedman. Caching Web Services in Mobile Ad-Hoc Networks: Opportunities and Challenges (2002).
- [11] Sam Clegg. Reducing the Network Overheads of .NET Remoting through Runtime Call Aggregation (2003).
- [12] W3C Web Services Activity (<http://www.w3.org/2002/ws/>)
- [13] Paper Appendix (<http://www.cs.usask.ca/~kae501/880/>)

Best Available Copy

How to Turn a GSM SIM into a Web Server *Projecting Mobile Trust onto the World Wide Web*

Scott Guthery, Roger Kehr, Joachim Posegga

Scott Guthery, Mobile-Mind, sguthery@mobile-mind.com

Roger Kehr, Deutsche Telekom Research, Roger.Kehr@Telekom.de

Joachim Posegga, Deutsche Telekom Research, Joachim.Posegga@Telekom.de

Key words: Smart Cards, GSM SIMs, Web Servers, Security Infrastructure

Abstract: We describe the WebSIM, an approach that integrates GSM SIMs into the Internet. The underlying idea is to implement a Web Server inside a SIM, and to allow for transparent access to it from the Internet.
The contribution of our approach is that a SIM, which is currently a security module (smart card) fitted in a GSM mobile phone, becomes also a personal security server in the Internet. Like any other server in the Internet, it speaks TCP/IP and processes HTTP requests, e.g. for accessing certain SIM services (e.g. authentication) via CGI scripts.
The Internet connectivity of a SIM inside a mobile phone can be achieved by having a proxy host tunnel IP packets to the SIM over SMS.

If we couldn't predict the Web, what good are we?
Bob Lucky, Vice President Bellcore, 1995

1. INTRODUCTION

Much of today's wireless Internet excitement is focused on the opportunities created by pushing the World Wide Web out onto mobile telephone networks. Micro-browsers and WAP handsets seek to turn the mobile telephone into a downscale laptop computer. Relatively little thought

has gone into wondering what the mobile telephone network can bring to the Web.

Secure, reliable authentication, which is a basic prerequisite for billing customers for services on a large scale, still has no globally-accepted solution. Various attempts have been made to provide the required security technology for the Internet, but none of them has widely succeeded so far. All approaches have in practice either been considered as too insecure, or too hard to establish at the user's side. With its strong similarity to the ubiquitous credit card, a smart card is a compelling component but the required infrastructure for smart card-based solutions has been found to be hard and costly to set up.

GSM, on the other hand, provides a widely used security infrastructure, in the form of symmetric keys distributed in subscriber identity modules (SIMs). More than 250 million GSM subscribers carry around these reduced size smart cards in their mobile phones. Mobile phones can thus be seen as "wireless card readers", with the add-on of providing an I/O channel to a user for applications running inside the SIM.

The theme of the current work is that while the Web is bringing its content to the mobile phone, the mobile phone can bring its trust to the Web. The idea is to provide the authentication and authorisation capabilities from the GSM SIM to Web-based applications in a Web-friendly way; viz. as a Web server. Such a WebSIM, like any other server in the Internet, speaks TCP/IP and is transparently accessible from Internet hosts via HTTP. Specific services offered by server-enabled SIMs, for example authentication, can be implemented on the SIM using CGI scripts.

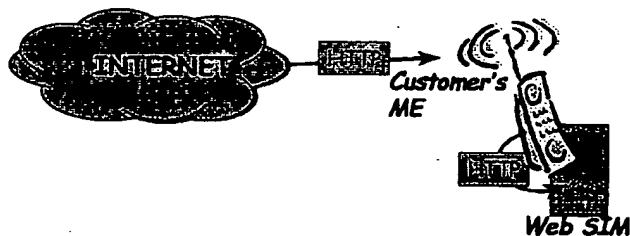


Figure 1. HTTP Requests to a SIM

Technically, this is achieved by implementing a Web server inside the GSM SIM and allowing for HTTP requests to this SIM and HTTP responses from it. Seen from the GSM perspective, this HTTP server provides selected parts of the existing application programming interface of a GSM SIM to the Internet. This makes communicating with a SIM residing in a mobile phone

identical to communicating with any Web server running in the Internet (cf. *Figure 1*), and the SIM can be transparently accessed from the Internet via HTTP, e.g. for authentication purposes.

2. THE WEBSIM: TECHNICAL DETAILS

A GSM SIM is an operator-trusted security server in GSM, performing computations on behalf of the GSM subscriber. The idea behind our approach, the WebSIM, is to extend the GSM SIMs into the Internet, by having them to understand a widely-used Internet protocol, HTTP, and to allow for transparent access over HTTP from the Internet. This means a GSM SIM residing in a mobile phone is not only a security server in the GSM network, but also becomes available as a security server in the World Wide Web, acting again on behalf of the GSM subscriber. We refer to such a SIM as a WebSIM.

A WebSIM, like any other server in the Internet, speaks TCP/IP and processes HTTP requests. Technically, this can be achieved by implementing a small, stripped-down Web server in a GSM SIM and making the SIMs accessible from the Internet. In this way, communicating with a SIM in a mobile phone, e.g. for authenticating a customer in the Internet, is the same as communicating with any Web server running in the Internet.

Seen from the GSM perspective, the idea behind the WebSIM is to make the interface of today's GSM SIMs (ETSI GSM 11.11 and GSM 11.14) partially available on the Internet.

2.1 The SIM's Web Server

Running a Web server in a SIM is less of a problem than one might think, in fact such servers for standard (non-GSM) smart cards were introduced in [Rees 99]. Clearly, a Web server in a SIM is not expected to host large amounts of information or HTML documents, but to provide a convenient interface to services of the SIM. These services, most of which will probably be security-related, can then be accessed via the standard protocol of the Web, HTTP, and implemented as server-side scripts on the SIM.

It is explicitly not an objective to implement versions of standard Internet services and protocols that are in full compliance with the specifications that define them. While fully compliant implementations of existing standards are certainly desirable, we are willing to give a little on full compliance, implementing only a strict subset of the specification, in order to realise an efficient yet useful smart card implementation. This design philosophy could

be summarised as "It's not how well the dog sings but that the dog sings at all."

A stripped-down version of the HTTP 1.0 protocol, which just covers the absolutely necessary part and only allow for one connection at a time, can be easily implemented with an application of less than 10K bytes inside the SIM. In particular, we can very elegantly implement this functionality as an applet on top of a GSM SIM Toolkit platform (ETSI GSM 02.19, 03.19) and then use the Toolkit's interpreter for server-side scripting. This also allows for interacting with the user of the SIM's mobile phone, since SIM Toolkit provides also an appropriate API for I/O (GSM 11.14).

2.2 Networking

Once we have an HTTP-server running in the SIM we need to connect it to the Internet. An elegant approach would be to see the mobile phone as a gateway router that passes IP packets to the SIM. If we do not want to assign a separate IP address to the SIM, we could also configure it as a process listening to port 80 on the mobile phone.

This approach, while technically elegant and easy to integrate into technologies like GPRS, requires modification of the handset and the creation of a new ETSI standard. Even if such a standard could be agreed to in a timely fashion, significant market penetration by compliant equipment would take at least two to three years. We therefore propose another approach, which can be implemented using today's protocols and mobiles.

The innovation barrier of the ME can be circumvented by a solution very common in the Internet; viz. a proxy server. We set up a proxy for the SIM on the Web and have this proxy tunnel HTTP packets through SMS to the SIM (cf. *Figure 2*). SMS messages arrive directly in the SIM and can be processed as required, e.g. by having Toolkit Applets register for such SMS. Thus, we circumvent the handset by using existing protocols and standards.

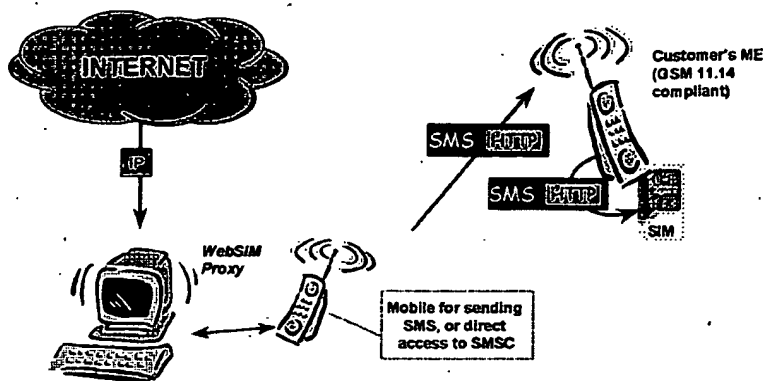


Figure 2. HTTP Tunnelling over SMS

The procedure for proxy-based IP-communication with the SIM over SMS is as follows:

1. An Internet host sends an HTTP request to the SIM's proxy.
2. The proxy embeds the request in a specially tagged SMS and sends it to the SIM.
3. The SIM passes the incoming SMS-encapsulated HTTP packet to the SIM that has registered to handle such tagged SMS.
4. The HTTP packet is extracted processed by the Web server in the SIM.
5. The HTTP response is embedded in SMS again and sent back to the proxy.
6. The proxy extracts HTTP response from the SMS and sends it back to the client in the Internet that sent the request.

As a result, the SIM can be transparently accessible by TCP/IP and HTTP from any Internet host. TCP/IP de-capsulation is handled by the proxy and the HTTP payload passed to the SIM in an SMS message. The response from the SIM is re-encapsulated by the proxy and returned to the Internet.

The proxy server approach also has some additional advantages since it can:

- implement a firewall between the Internet and the GSM network.
- guard against denial of service attacks.
- perform address translation (NAT) between the Internet address of a SIM card and the GSM address of the handset which holds the SIM.

- perform accounting and billing for WebSIM services
- eliminate the need for implementing a TCP/IP stack in the SIM

When used in practice, the WebSIM processes HTTP-bearing IP packets, for example, an URL request such as

`http://websim.dtrd.de/+49000000000/sign=(2A49C01...)`

This HTTP request can initiate the signing of the data in brackets in the SIM of the named GSM phone. After processing the request, which might consist of running other SIM-internal applications or commands, the result is sent back to the originating Internet host. Thus, integrating the capabilities of the GSM SIM into Internet applications is just like communication with any other Web server on the Internet.

2.3 Implementation

We have implemented a prototype of a WebSIM and its proxy that allows access a few SIM services over HTTP. The proxy's name is websim.dtrd.de¹, the SIM is identified by its phone number. The current implementation provides access to the following services:

- a) `http://websim.dtrd.de/+49000000000/info`
Returns information about LAI and LAC of the SIM (GSM 11.14 PROVIDE LOCAL INFO)
- b) `http://websim.dtrd.de/+49000000000/si=(item1,item2,item3,...)`
Prompts the user of the phone with a GSM 11.14 SELECT ITEM command offering the choices listed in brackets, separated by ",". Each item is interpreted as a string, the overall length of all strings must not exceed 120 bytes. The user's choice is returned.
- c) `http://websim.dtrd.de/+49000000000/gi=(prompt)`
Runs the GSM 11.14 GET INPUT command and returns the text that has been entered.
- d) `http://websim.dtrd.de/+49000000000/dt=(text)`
Runs the GSM 11.14 DISPLAY TEXT with the argument supplied.

¹ Access to the server is restricted.

- e) `http://websim.dtrd.de/+49000000000/sign=(abcdef0123456789)`
Encrypts the argument (interpreted as a string of hexadecimal characters) and returns the result.

The length of the arguments (given in brackets) is, for the sake of simplicity, restricted to fit into one SMS.

2.3.1 Proxy Implementation

The proxy is a Linux laptop running an Apache Web server with a couple of CGI scripts. These CGI scripts (implemented in Perl) take an incoming HTTP request, embed it in an SMS message, and send it off to the specified phone number.

Sending of SMS is done through a GSM mobile that is attached to the laptop with a PCMCIA modem card, which is used for sending and receiving SMS². Short messages are sent by turning the modem into TPDU mode (GSM 07.05), using a tag that causes the message to go directly to the Web Server application in the destination SIM (cf. GSM 03.48 and GSM 11.14).

Receiving SMS messages is detected by a separate process looping on the laptop that continuously polls the attached mobile for incoming short messages which are responses to pending HTTP requests. If an incoming short message is detected, it is fetched, the HTTP response is extracted and TCP/IP encapsulated and returned to the Internet client that sent the corresponding request.

2.3.2 Web Server Applet

The Web server runs as an applet on the SIM Toolkit platform (GSM 03.19) in a Schlumberger Simera SIM. The applet is written in Java, its size is currently about 7K bytes of Java byte codes. For the sake of simplicity the applet makes the following restrictions:

- a) an HTTP-Request must not exceed one SMS, and one SMS can only contain one request.
- b) the card handles only one request at a time, i.e. there is no session management inside the card.

Both restrictions can be easily overcome if needed.

We did not space-optimize the applet code at all; and we believe that it can be stripped down to a size of about 5K bytes. Noteworthy, adding new commands to the server Applet does not significantly increase its size: HTTP

² A more efficient variant would be to connect the proxy directly to the network's short message service centre (SMSC) which is the store-and-forward point for all SMS messages.

can be seen as a general-purpose application launching protocol, and once the basic HTTP-handling functionality is implemented, adding an extra command increases the applet only slightly: the difference between an applet providing only the SELECT ITEM command, and the version handling the five commands above is only a few hundred bytes.

2.3.3 Example Request

The HTTP-request `http://websim.dtrd.de/+49000000000/info` results in the following response (without HTTP headers):

LAI: 262 01
LAC: 730C

262 is the country code (Germany), 01 denotes the network (D1-Telekom), and 730C is the Local Area Code (Karlsruhe, Germany).

Overall processing time depends largely on SMS transport time, which is usually between 5 and 20 seconds one way. The proxy needs 3-4 seconds to send and receive the short messages, the handset and SIM-internal processing takes roughly another 5-6 seconds. So, the complete processing of such an HTTP request takes typically about 30 seconds.

3. APPLICATIONS OF THE WEBSIM

The WebSIM is not an application per se: it opens up the SIM to the Internet and provides an Internet-compliant interface to SIM services. Thus, it is a horizontal technology (more precisely a middleware) that supports "dot com" style applications.

We sketch a few of these applications for different domains below. Much more is possible, and in fact the most promising aspect is that the WebSIM a very convenient middleware for integrating it into Internet applications. Rather than having to deal with a different interface to a SIM each time, Internet applications can access and activate these applets in their own language, HTTP.

3.1 Provision of a secure I/O channel for Electronic commerce

Assume a customer of an Internet bookshop ordered a book for US\$ 20. When ordering it, the phone number of a WebSIM phone was provided, and the Internet bookshop can now launch a simple HTTP-request such as

[http://www.../+49000.../si=\(***Bookshop:,Confirm%20USD20",Cancel,...\)](http://www.../+49000.../si=(***Bookshop:,Confirm%20USD20)

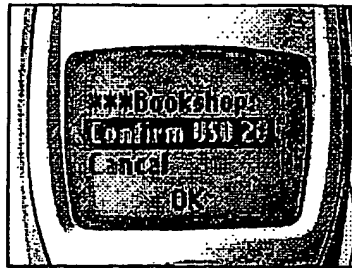


Figure 3. Screen Shot

This would result in a menu being displayed on the mobile phone of the customer (cf. *Figure 3*), and the Internet merchant would have established a relatively secure I/O channel to its customer (or whoever is using the customer's phone).

The security of this I/O channel can easily be enhanced by connecting over HTTPS to the proxy, or by cryptographic means as we will see next.

3.2 Authentication on the Internet

Assume Alice wants to authenticate Bob over the Internet, Oscar is Bob's GSM operator that issued a WebSIM SIM_B to Bob. Consider the following basic skeleton of a protocol:

1. B \rightarrow A: +4900000000 [Bob's telephone number]
2. A \rightarrow SIM_B : [http://www.oscar.com/+4900000000/sign\(RAND\)](http://www.oscar.com/+4900000000/sign(RAND))
3. $SIM_B \rightarrow$ A: $\text{hash}(K_i; \text{RAND}, \text{IMSI})$
4. A \rightarrow O: [https://www.oscar.com/verify\(RAND, hash\(K_i; RAND, IMSI\), +4900000000\)](https://www.oscar.com/verify(RAND, hash(K_i; RAND, IMSI), +4900000000))
5. O \rightarrow A: yes/no

In step 1 Bob gives Alice the phone number of his SIM. Alice then sends a random number (challenge) to the SIM_B in step 2. The SIM_B returns a hash of a secret key K_i , the random number RAND, and the SIM's IMSI³ to Alice in step 3.

³ IMSI = International Mobile Subscriber Identity number.

Alice can now send the response she got in step 3 to Oscar the operator, who can verify the result. Oscar knows the hash algorithm, the secret key K_i , and he can associate the phone number with the IMSI that was used in the encryption. Note that nobody besides Oscar needs to know K_i and hash. Note also that the messages of step 4 and 5 should be sent over a secure channel, e.g. by using `https://...` between Alice and Oscar.

Such a protocol can be easily refined to meet various needs one has for authentication, like including an explicit conformation from Bob, or a time stamp (for instance taken from the SMS), etc. It is a classical challenge/response authentication which can be applied to many scenarios (home banking, access control, etc.), or it can be easily adapted to provide, for instance, a session key for other purposes. For security reasons, the scenario can also be used with keys other than K_i , or with a key derived from K_i .

Essentially, this scenario is based on the principle that the mobile phone can be used as a "wireless" card reader containing a authentication token, and the security infrastructure of GSM can be easily accessed from the Internet.

3.3 Physical Access

Another nice example for using a WebSIM is the following one. Assume pushing your doorbell at home results in a request such as:

`http://websim.dtrd.de/+49000000/si=(Open,Call Intercom,Cancel)`

If you are standing in front of your door and have pushed the doorbell button, you will of course select "Open" on the menu appearing on your phone. If not, you can select to be connected with your home's internal intercom or simply cancel the request if you don't want to be disturbed.

In the case that you are connected to the intercom (which is in turn bridged to your mobile handset) you can converse with your visitor and if the situation warrants it, open the door remotely even if you aren't at home.

3.4 Handset Configuration

There are a number of SIM services local to the mobile handset that would be much easier to handle with Web interfaces. For example, the management of the SIM-internal phone book, which can be very

conveniently, updated using a Web browser if the WebSIM understands HTTP POST-methods.

4. CONCLUSION

We have presented the WebSIM, an approach integrates the GSM security module (SIM) into the Internet. The underlying idea is to integrate an HTTP server into a GSM SIM, allowing Internet connections to be made to it. This turns the SIM, which is security server for the GSM subscriber, into a WebSIM, which is a general-purpose security server for the subscriber on the Internet.

Such a WebSIM, like any other server in the Internet is transparently accessible from Internet hosts via TCP/IP and HTTP. Specific services offered by SIMs, e.g. authentication, can be accessed using CGI scripts from Internet hosts.

Seen from the GSM perspective, this HTTP server extends parts of the existing external interface of today's GSM SIMs into the Internet and is seamlessly reachable from the Internet, acting as a security server for a GSM subscriber.

The main contribution of our approach is to provide the function of SIMs in Internet-compliant protocols anyone can use. This means that the barriers for smart card applications today, 1) the lack of integration of smart cards into information technology architectures and the World Wide Web and 2) the complex interaction with cards using APDUs, is overcome by providing a simple, standard protocol, i.e. HTTP, for accessing SIM services.

A WebSIM can be accessed from anywhere on the Web using familiar Web protocols, and application programmers do not need to cope with smart card-specific interfaces any more. As with other servers in the Internet, the WebSIM processes HTTP-bearing IP packets. For example, an URL request such as

`http://websim.dtrd.de/+491710000000/sign=2A49C01`

would initiate the identity application running in the WebSIM of the named GSM phone.

After processing the request, which might consist of running other SIM-internal applications or commands, the result is sent back to the originating Internet host. Thus, integrating SIM-based security into Internet applications involves the same programming techniques as communicating with any other Web server on the Internet.

Sales of mobile handsets are forecast to soon exceed sales of personal computers. While desktop and laptop computers are great for viewing art at the Louvre, when it comes to taking action, the mobile handset with its trust-bearing SIM not to mention its portability may prove to be the primary Web surfing device. WebSIM enables the mobile to add value to the Web as opposed to simply receiving content from it.

5. REFERENCES

- [Barber 99] J. Barber, "The Smart Card URL Programming Interface," Gemplus Developer Conference, June, 21 - 22, CNIT Conference Center, Paris-La Defense, France. 1999.
- [Becker 99] C.B. Becker, B. Patil and E. Qaddoura, IP Mobility Architecture Framework, IETF-Draft, October, 1999.
- [Campbell 99] A. Campbell, J. Gomez, C-Y. Wan, Z. Turanyi, and A. Valko, Cellular IP, IETF-Draft, October, 1999.
- [Comer 95] D. Comer, Internetworking with TCP/IP, Prentice Hall, 1995.
- [Di Giorgio 99] Rinaldo Di Giorgio, An introduction to the URL programming interface, Java World, September 1999.
- [GSM 02.19] Digital cellular telecommunications system (Phase 2+, Release 98): Subscriber Identity Module Application Programming Interface (SIM API); Service description; Stage 2. European Telecommunications Standards Institute, Sophia Antipolis, France. Unpublished Draft. 1999.
- [GSM 03.19] Digital cellular telecommunications system (Phase 2+); Subscriber Identity Module Application Programming Interface (SIM API); SIM API for Java Card™; Stage 2. European Telecommunications Standards Institute, Sophia Antipolis, France. Unpublished Draft. 1999.
- [GSM 03.60] Digital cellular telecommunications system (Phase 2+); General Packet Radio Service (GPRS); Service description; Stage 2. European Telecommunications Standards Institute, Sophia Antipolis, France. Available from <http://www.etsi.org/>. 1999.
- [GSM 11.11] European digital cellular telecommunications system (Phase 2); Specification of the Subscriber Identity Module - Mobile Equipment (SIM-ME) interface (GSM 11.11). European Telecommunications Standards Institute, Sophia Antipolis, France. Available from <http://www.etsi.org/>. 1998.
- [GSM 11.14] European Digital cellular telecommunications system (Phase 2+): Specification of the SIM application toolkit for the Subscriber Identity Module-Mobile Equipment (SIM-ME) interface (GSM 11.14). European Telecommunications Standards Institute, Sophia Antipolis, France. Available from <http://www.etsi.org/>. 1998.

[Gustafsson 99] E. Gustafsson, A. Jonsson, E. Hubbard, J. Malmkvist, Requirements on Mobile IP from a Cellular Perspective, IETF-Draft, June, 1999.

[Guthery 00] S. Guthery, J. Posegga, Y. Baudoin, J. Rees, "IP and ARP over ISO 7816-3", IETF Internet-Draft, February, 1, 2000.

[Rees 99] Jim Rees and Peter Honeyman: Webcard: a Java Card web server. CITI Technical Report 99-3, Center for Information Technology Integration, University of Michigan. October 1999. [http://www.citi.umich.edu/projects/sinciti/smart card/webcard/citi-tr-99-3.html](http://www.citi.umich.edu/projects/sinciti/smart%20card/webcard/citi-tr-99-3.html)

[Reichenbach 98] Martin Reichenbach, Herbert Damker, Hannes Federrath, Kai Rannenberg: Individual Management of Personal Reachability in Mobile Communication; pp. 163-174 in Louise Yngström, Jan Carlsen: Information Security in Research and Business; Proceedings of the IFIP TC11 13th International Information Security Conference (SEC '97): 14-16 May 1997, Copenhagen, Denmark, Chapman & Hall, London, 1998 Herbert Damker, Ulrich Pordesch, Kai Rannenberg, Michael Schneider: Aushandlung mehrseitiger Sicherheit: der Erreichbarkeits- und Sicherheitsmanager, In: Günter Müller, Kurt-Hermann Stapf: Mehrseitige Sicherheit in der Kommunikationstechnik - Erwartung, Akzeptanz, Nutzung; Addison-Wesley-Longman, 1998.

[RFC 1945] T. Berners-Lee, R. Fielding, and H. Frystyk. Hypertext Transfer Protocol – HTTP/1.0, IETF RFC 1945, May, 1996.

[RFC 2396] T. Berners-Lee, R. Fielding, and L. Masinter. Uniform Resource Identifiers (URI): Generic Syntax, IETF RFC 2396, August, 1998.

[RFC 2616] R. Fielding, J. Gettys, J. Mogul, H. Frystyk, L. Masinter, Leach, and T. Berners-Lee: Hypertext Transfer Protocol – HTTP/1.1, June, 1999

[Vaha-Sipila 99] A. Vaha-Sipila, URLs for GSM Short Message Service, IETF-Draft, May 19, 1999.

This Page is inserted by IFW Indexing and Scanning
Operations and is not part of the Official Record

BEST AVAILABLE IMAGES

Defective images within this document are accurate representations of the
original documents submitted by the applicant.

Defects in the images include but are not limited to the items checked:

- ☐ BLACK BORDERS
- ☐ IMAGE CUT OFF AT TOP, BOTTOM OR SIDES
- ☐ FADED TEXT OR DRAWING
- ☒ BLURED OR ILLEGIBLE TEXT OR DRAWING
- ☐ SKEWED/SLANTED IMAGES
- ☐ COLORED OR BLACK AND WHITE PHOTOGRAPHS
- ☐ GRAY SCALE DOCUMENTS
- ☐ LINES OR MARKS ON ORIGINAL DOCUMENT
- ☐ REPERENCE(S) OR EXHIBIT(S) SUBMITTED ARE POOR QUALITY
- ☐ OTHER: _____

IMAGES ARE BEST AVAILABLE COPY.

**As rescanning documents *will not* correct images
problems checked, please do not report the
problems to the IFW Image Problem Mailbox**

09/912,636

TV

**A Database of Computer Attacks for the Evaluation of Intrusion
Detection Systems**

by

Kristopher Kendall

Submitted to the Department of Electrical Engineering and Computer Science in partial
fulfillment of the requirements for the degrees of

**Bachelor of Science in Computer Science and Engineering
and Master of Engineering in Electrical Engineering and Computer Science**

at the

MASSACHUSETTS INSTITUTE OF TECHNOLOGY

June 1999

© Kristopher Kendall, MCMXCVIV. All rights reserved.

The author hereby grants to MIT permission to reproduce and distribute publicly paper
and electronic copies of this thesis document in whole or in part, and to grant others the
right to do so.

Author
Department of Electrical Engineering and Computer Science,
May 21, 1999

Certified by.....
Richard Lippmann
Senior Scientist, MIT Lincoln Laboratory
Thesis Supervisor

Accepted by.....
Arthur C. Smith
Chairman, Department Committee on Graduate Theses

*This work was sponsored by the Department of Defense Advanced Research Projects Agency
(DARPA). Opinions, interpretations, conclusions, and recommendations are those of the author
and are not necessarily endorsed by DARPA.

A Database of Computer Attacks for the Evaluation of Intrusion Detection Systems

by
Kristopher Kendall

Submitted to the Department of Electrical Engineering and Computer Science

May 21, 1999

In Partial Fulfillment of the Requirements for the Degree of
Bachelor of Science in Computer Science and Engineering
and Master of Engineering in Electrical Engineering and Computer Science

Abstract

The 1998 DARPA intrusion detection evaluation created the first standard corpus for evaluating computer intrusion detection systems. This corpus was designed to evaluate both false alarm rates and detection rates of intrusion detection systems using many types of both known and new attacks embedded in a large amount of normal background traffic. The corpus was collected from a simulation network that was used to automatically generate realistic traffic—including attempted attacks.

The focus of this thesis is the attacks that were developed for use in the 1998 DARPA intrusion detection evaluation. In all, over 300 attacks were included in the 9 weeks of data collected for the evaluation. These 300 attacks were drawn from 32 different attack types and 7 different attack scenarios. The attack types covered the different classes of computer attacks and included older, well-known attacks, newer attacks that have recently been released to publicly available forums, and some novel attacks developed specifically for this evaluation.

The development of a high quality corpus for evaluating intrusion detection systems required not only a variety of attack types, but also required realistic variance in the methods used by the attacker. The attacks included in the 1998 DARPA intrusion detection evaluation were developed to provide a reasonable amount of such variance in attacker methods. Some attacks occur in a single session with all actions occurring in the clear, while others are broken up into several sessions spread out over a long period of time with the attacker taking deliberate steps to minimize the chances of detection by a human administrator or an intrusion detection system. In some attacks, the attacker breaks into a computer system just for fun, while in others the attacker is interested in collecting confidential information or causing damage. In addition to providing detailed descriptions of each attack type, this thesis also describes the methods of stealthiness and the attack scenarios that were developed to provide a better simulation of realistic computer attacks.

Thesis Supervisor: Richard Lippmann
Title: Senior Scientist, MIT Lincoln Laboratory

Acknowledgments

First, I would like to thank my thesis advisor, Rich Lippmann, for keeping my thesis on track and for providing me with many valuable ideas and insights. I would also like to thank Rob Cunningham and Dave Fried for taking the time to read and comment on early drafts of this thesis. I appreciate the support of Rich, Rob, Dave, Isaac Graf, Seth Webster, Dan Weber, and Jonathon Korba, and the rest of the intrusion detection evaluation staff, who made working in this group a fun and highly educational experience. Finally, I would like to thank Debbie and my parents, whose love and support fuel all of my endeavors.

Contents

Chapter 1 Introduction	8
1.1 1998 DARPA Intrusion Detection System Evaluation.....	8
1.2 The Development of Attacks for the 1998 DARPA Evaluation.....	10
1.3 Outline of the Thesis.....	10
Chapter 2 Background	12
2.1 Overview of Computer Attacks.....	12
2.2 Intrusion Detection Systems.....	13
2.3 Strategies for Intrusion Detection.....	15
Chapter 3 Simulation Network	20
3.1 Modeling an Air Force Local Area Network.....	20
3.2 Simulation Hardware and Network Topology.....	21
3.3 Simulation Software.....	23
3.3.1 Virtual Machines.....	23
3.3.2 Traffic Generation.....	24
Chapter 4 Exploits	26
4.1 Sources.....	26
4.2 Age of an Exploit.....	27
4.3 A Taxonomy for Computer Attacks.....	28
4.3.1 Privilege Levels.....	29
4.3.2 Methods of Transition or Exploitation.....	30
4.3.3 Transitions Between Privilege Levels.....	31
4.3.4 Actions.....	31
4.3.5 Using the Taxonomy to Describe Attacks.....	34
4.3.6 Examples.....	35
Chapter 5 Exploits for the 1998 DARPA Evaluation	37
Chapter 6 Denial of Service Attacks	39
6.1 Apache2 R-a-Deny(Temporary/Administrative).....	40
6.2 Back R-a-Deny(Temporary).....	42

6.3	Land R-b-Deny(Administrative).....	43
6.4	Mailbomb R-a-Deny(Administrative)	44
6.5	SYN Flood (Neptune) R-a-Deny(Temporary).....	45
6.6	Ping Of Death R-b-Deny(Temporary).....	47
6.7	Process. Table R-a-Deny(Temporary).....	48
6.8	Smurf R-a-Deny(Temporary)	51
6.9	Syslogd R-b-Deny(Administrative).....	54
6.10	Teardrop R-a-Deny(Temporary).....	55
6.11	Udpstorm R-a-Deny(Administrative).....	56
Chapter 7 User to Root Attacks		58
7.1	Eject U-b-S	60
7.2	Ffbconfig U-b-S	63
7.3	Fdformat U-b-S.....	64
7.4	Loadmodule U-b-S	65
7.5	Perl U-b-S	67
7.6	Ps U-b-S.....	68
7.7	Xterm U-b-S	69
Chapter 8 Remote to User Attacks		70
8.1	Dictionary R-a-U	70
8.2	Ftp-write R-c-U.....	74
8.3	Guest R-c-U.....	75
8.4	Imap R-b-S.....	76
8.5	Named R-b-S	77
8.6	Phf R-b-U.....	78
8.7	Sendmail R-b-S.....	79
8.8	Xlock R-cs-Intecept(Keystrokes)	81
8.9	Xsnoop R-c-Intecept(Keystrokes)	83
Chapter 9 Probes		85
9.1	Ipsweep R-a-Probe(Machines)	86
9.2	Mscan R-a-Probe(Known Vulnerabilities)	87
9.3	Nmap R-a-Probe(Services).....	88
9.4	Saint R-a-Probe(Known Vulnerabilities).....	90

9.5	Satan R-a-Probe(Known Vulnerabilities)	93
Chapter 10 Realistic Intrusion Scenarios		95
10.1	Attack Scenarios	95
10.1.1	Cracker	96
10.1.2	Spy.....	96
10.1.3	Rootkit.....	96
10.1.4	Http Tunnel	97
10.1.5	SNMP Monitoring.....	98
10.1.6	Multihop.....	98
10.1.7	Disgruntled/Malicious User	99
Chapter 11 Stealthiness and Actions		100
11.1	Avoiding Detection of Denial of Service (R-Deny)	100
11.2	Avoiding Detection of Probes (R-Probe).....	101
11.3	Avoiding Detection of User to Root (L-?-S) Attacks	102
11.4	Avoiding Detection of Remote to Local (R-?-L) Attacks	106
11.5	Actions	107
Chapter 12 Attack Planning and Data Collection		109
12.1	Planning and Keeping Track of Attacks	109
12.2	Numbers of Attacks	110
12.3	Verification of Attack Success	111
Chapter 13 Results and Future Work		113
13.1	Results of the 1998 Evaluation	113
13.2	Future Work.....	116
13.2.1	Inclusion of Windows NT and Other Systems in the Evaluation.....	116
13.2.2	Better Automation of Attack Planning and Verification.....	116
13.2.3	Improved Stealthy Attacks	117
Appendix A Attack Schedule for Test Dataset		118

List of Figures

2-1	Approches to Intrusion Detection.....	16
3-1	Simulation Network Topology.....	22
4-1	Vulnerability Decreases as Time Passes.....	27
4-2	Summary of Possible Types of Actions.....	32
4-3	A Summary of Possible Attack Descriptions.....	34
5-1	The Attacks Used in the 1998 DARPA Intrusion Detection Evaluation.....	37
6-1	Summary of Denial of Service Attacks.....	40
6-2	The Internet is Used to Amplify a Ping Flood and Create a Smurf Attack.....	52
6-3	UDPStorm is Triggered by a Single Spoofed Packet.....	57
7-1	Summary of User to Root Attacks.....	59
7-2	C Code for the Eject Exploit.....	61
8-1	Summary of Remote to Local Attacks.....	71
8-2	Plot of Connections to Pop3 Service During a Dictionary Attack.....	72
8-3	Illustration of Sendmail Attack.....	80
9-1	Summary of Probes.....	85
9-2	Plot of Connections During a Medium Level Saint Scan.....	91
9-3	Plot of Connections During a Medium Level Satan Scan.....	94
10-1	Attacker Uses Http to Tunnel Through a Firewall.....	98
11-1	Attacker Can Use "at" to Temporally Disassociate the Time of Access and Time of Attack.....	104
11-2	Transcript of a Stealthy Eject Attack.....	105
12-1	Instances of Non-Scenario Attacks.....	111
13-1	Current Intrusion Detection Systems Cannot Find New DoS and Remote to User Attacks.....	114
13-2	Clear vs Stealthy Attack Detection Rates.....	115

Chapter 1

Introduction

1.1 1998 DARPA Intrusion Detection System Evaluation

Heavy reliance on networked computer resources and the increasing connectivity of these networks has greatly increased the potential damage that can be caused by attacks launched against computers from remote sources. These attacks are difficult to prevent with firewalls, security policies, or other mechanisms because system and application software is changing at a rapid pace, and this rapid pace often leads to software that contains unknown weaknesses or bugs. Intrusion detection systems are designed to detect those attacks that inevitably occur despite security precautions. Some intrusion detection systems detect attacks in real time and can be used to stop an attack in progress. Others provide after-the-fact information about attacks that can be used to repair damage, understand the attack mechanism, and reduce the possibility of future attacks of the same type [43].

Many parties are working on the development of intrusion detection systems, including universities, commercial software companies, and organizations within the Department of Defense. As these groups explore different methods and develop various new systems for intrusion detection, it is clearly advantageous to have a means of evaluating the success of these systems in detecting attacks. The best environment for

testing and evaluation of an intrusion detection system is the actual environment in which it will be used. However, research groups often do not have access to operational networks on which to test their systems, and these systems (especially while they are still in early development) are tested in a simulated environment. The ability to perform accurate testing and evaluation in a simulated environment requires high-quality data that is similar to the traffic (including attacks) that one finds on operational networks. In general, this data is difficult to acquire because it contains private information and reveals potential vulnerabilities of the networks from which the data is collected. These factors led to DARPA sponsorship of MIT Lincoln Laboratory's 1998 intrusion detection evaluation, which created the first standard corpus for the evaluation of intrusion detection systems.

The 1998 intrusion detection evaluation was the first of an ongoing series of yearly evaluations conducted by MIT Lincoln Laboratory under DARPA ITO and Air Force Research Laboratory sponsorship. These evaluations contribute significantly to the intrusion detection research field by providing direction for research efforts and calibration of current technical capabilities. The 1998 evaluation was designed to be simple, to focus on core technology issues, and to encourage the widest possible participation by eliminating security and privacy concerns and by providing data types that are used by the majority of intrusion detection systems. Data for the first evaluation was made available in the summer of 1998. The evaluation itself occurred towards the end of the summer. A follow-up meeting for evaluation participants and other interested parties was held in December 1998 to discuss the results of the evaluation.

1.2 The Development of Attacks for the 1998 DARPA Evaluation

This thesis describes the computer attacks that were included in the 1998 DARPA intrusion detection evaluation. A large sample of actual computer attacks was needed to accurately test the performance of intrusion detection systems. These attacks needed to cover the different classes of attack types. Many of the attacks used in the evaluation were drawn from public sources, but some novel attacks were developed specifically for use in this evaluation. In all cases, these attacks had to be adapted to work reliably in the largely automated simulation network from which the 1998 DARPA evaluation data were collected. Later sections of this thesis discuss the methods that were developed to create realistic simulations of computer intrusion scenarios, and the methods that were developed to vary the degree of attack stealthiness. People who attack computer networks often have goals beyond simply gaining access to a system. Some attackers break into computers simply for the challenge, others are interested in collecting information, and some are motivated by the desire to cause damage. Attackers also vary in their level of sophistication, and an accurate evaluation of intrusion detection systems requires testing how well the systems are able to detect attacks from all types of attackers—from the relative novice who is not aware that an intrusion detection system is monitoring a network, to the sophisticated, experienced cracker who knows about intrusion detection systems and takes steps to avoid being caught.

1.3 Outline of the Thesis

Chapter 2 presents background information about computer attacks and intrusion detection systems. This provides perspective for later discussion on the development of

attacks for use in evaluating these systems. The different types of computer attacks and some of the many intrusion detection strategies that are currently being developed are discussed.

Chapter 3 describes the simulation network that was used to collect data for the 1998 DARPA intrusion detection evaluation. This network consisted of 11 computers and one router that, with the aid of software developed for use in the evaluation, simulated a large network consisting of hundreds of machines, and thousands of users.

Chapters 4, 5, 6, 7, 8, and 9 focus on the different types of attacks that were developed for use in the evaluation. Chapter 4 presents a taxonomy of computer attacks that was useful in choosing attack types to include in the evaluation. Chapter 5 is an introduction to Chapters 6 through 9 which discuss in detail the specific attacks within each of the four broad categories of attacks included in the evaluation (denial of service attacks, attacks that give a local user superuser access, attacks that give a remote user local access, and probes).

Chapters 10 and 11 discuss the attack scenarios and stealthy methods that were used to create a more realistic simulation of actual computer attacks.

Chapter 12 describes the processes that were used to plan, collect, and verify the attack instances that were included in the 1998 DARPA intrusion detection evaluation.

Finally, in Chapter 13 the results of the 1998 DARPA intrusion detection evaluation are summarized and discussed with a focus on the implications of these results for both the intrusion detection research community, and the Lincoln Laboratory group that will be conducting future evaluations.

Chapter 2

Background

2.1 Overview of Computer Attacks

In its broadest definition, a computer attack is any malicious activity directed at a computer system or the services it provides. Examples of computer attacks are viruses, use of a system by an unauthorized individual, denial-of-service by exploitation of a bug or abuse of a feature, probing of a system to gather information, or a physical attack against computer hardware. A subset of the possible types of computer attacks were included in the 1998 DARPA intrusion detection system evaluation, including: (1) Attacks that allow an intruder to operate on a system with more privileges than are allowed by the system security policy, (2) Attacks that deny someone else access to some service that a system provides, or (3) Attempts to probe a system to find potential weaknesses.

The following paragraphs provide some examples of the many ways that an attacker can either gain access to a system or deny legitimate access by others.

Social Engineering: An attacker can gain access to a system by fooling an authorized user into providing information that can be used to break into a system. For example, an attacker can call an individual on the telephone impersonating a network administrator in an attempt to convince the individual to reveal confidential

information (passwords, file names, details about security policies). Or an attacker can deliver a piece of software to a user of a system which is actually a trojan horse containing malicious code that gives the attacker system access.

Implementation Bug: Bugs in trusted programs can be exploited by an attacker to gain unauthorized access to a computer system. Specific examples of implementation bugs are buffer overflows, race conditions, and mishandled of temporary files.

Abuse of Feature: There are legitimate actions that one can perform that when taken to the extreme can lead to system failure. Examples include opening hundreds of telnet connections to a machine to fill its process table, or filling up a mail spool with junk e-mail.

System Misconfiguration: An attacker can gain access because of an error in the configuration of a system. For example, the default configuration of some systems includes a "guest" account that is not protected with a password.

Masquerading: In some cases it is possible to fool a system into giving access by misrepresenting oneself. An example is sending a TCP packet that has a forged source address that makes the packet appear to come from a trusted host.

2.2 Intrusion Detection Systems

Intrusion detection systems gather information from a computer or network of computers and attempt to detect intruders or system abuse. Generally, an intrusion detection system will notify a human analyst of a possible intrusion and take no further action, but some newer systems take active steps to stop an intruder at the time of detection [49].

Although there are many possible sources of data an intrusion detection system can use, three types of data were provided to participants in the 1998 Lincoln Laboratory intrusion detection evaluation. Most intrusion detection systems in existence today use one or more of these three types of data. The first of these data sources is traffic sent over the network. All data that is transmitted over an ethernet network is visible to any machine that is present on the local network segment. Because this data is visible to every machine on the network, one machine connected to this ethernet can be used to monitor traffic for all the hosts on the network. During the DARPA evaluation, network traffic was sniffed using a single machine running the tcpdump program [39] to save the network traffic. A second source of data for an intrusion detection system is system-level audit data. Most operating systems offer some level of auditing of operating system events. The amount of data that is collected could be as limited as logging failed attempts to log in, or as verbose as logging every system call. Basic Security Module (BSM) [62] data from a Solaris victim machine was collected and distributed as part of the DARPA evaluation data. A third source of data distributed to the evaluation participants was information about file system state. Daily file system dumps were collected from each of the machines used in the simulation. An intrusion detection system that examines this file system data can alert an administrator whenever a system binary file (such as the ps, login, or ls program) is modified. Normal users have no legitimate reason to alter these files, so a change to a system binary file indicates that the system has been compromised. Although there are many other potential sources of data that can be used by an intrusion detection system to find attacks (such as real-time process lists, logfiles, processor loads, etc.), these three sources (sniffed network traffic,

host-level audit files, and file-system state) were provided to participants in the 1998 Lincoln Laboratory DARPA intrusion detection evaluation because they were determined to be the sources most commonly used by the evaluation participants.

After the three types of data were collected and aggregated, the data was distributed to participants via CD-ROM. Once participants obtained this data, each group used its particular intrusion detection system to find intrusions and abuses that were inserted into the collected traffic. Although the 1998 DARPA evaluation tested only the ability to find attacks offline, some intrusion detection systems can evaluate data in real-time, allowing administrators (or the system itself) to take defensive action against the intruder.

2.3 Strategies for Intrusion Detection

The different approaches that have been pursued to develop intrusion detection systems are described in many papers, including [3][44][63]. Figure 2-1 shows four major approaches to intrusion detection and the different characteristics of these approaches. The lower part of this figure shows approaches that detect only known attacks, while the upper part shows approaches that detect novel attacks. Simpler approaches are shown on the left and approaches that are both computationally more complex and have greater memory requirements are shown towards the right.

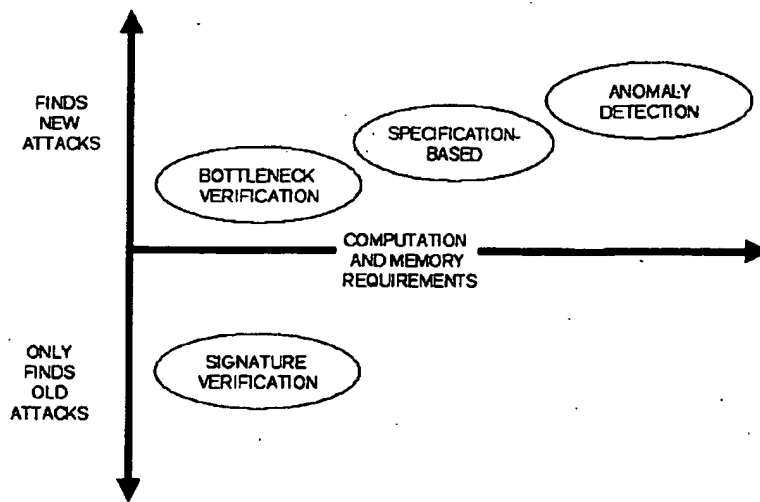


Figure 2-1: Approaches to Intrusion Detection

The most common approach to intrusion detection, denoted as “signature verification” is shown on the bottom of Figure 2-1. Signature verification schemes look for an invariant sequence of events that match a known type of attack. For example, a signature verification system that is looking for a Ping Of Death denial-of-service attack (an oversize ping packet that causes some machines to reboot) would have a simple rule that says “any ping packet of length greater than 64 kilobytes is an attack.” Attack signatures can be devised that detect attempts to exploit many possible system vulnerabilities, but a large drawback of this strategy is that it is difficult to establish rules that identify novel types of attacks. The Network Security Monitor (NSM) was an early signature-based intrusion detection system that found attacks by searching for keywords in network traffic captured using a sniffer. Early versions of the NSM [29][28] were the foundation of many government and commercial intrusion detection systems, including

NetRanger [23] and NID [41]. Signature verification systems are popular because one sniffer can monitor traffic to many workstations and the computation required to reconstruct network sessions and search for keywords is not excessive. In practice, these systems can have high false-alarm rates (e.g. 100's of false-alarms per day) because it is often difficult to select keywords by hand that successfully detect real attacks without creating false alarms for normal traffic. In addition, signature verification schemes must be updated frequently to detect new attacks as they are discovered. Recent research on systems which rely on signature verification includes BRO [47] and NSTAT [38].

The approaches shown in the upper half of Figure 2-1 can be used to find novel attacks. This capability is essential to protect critical hosts because new attacks and attack variants are constantly being developed.

Anomaly detection, shown in the upper right of Figure 2-1, is one of the most frequently suggested approaches to detect novel new attacks. Anomaly detection schemes construct statistical models of the typical behavior of a system and issue warnings when they observe actions that deviate significantly from those models. NIDES was one of the first statistical-based anomaly detection systems used to detect unusual user [36] and unusual program [1] behavior. The statistical component of NIDES forms a model of a user, system, or network activity during an initial training phase. After training, anomalies are detected and flagged as attacks. Of course, anomalous behavior does not always signal that an attack is taking place, so anomaly detection systems need to be carefully tuned to avoid high false alarm rates. This level of tuning is only possible if normal user or system activity is stable over time and does not overlap with attacker activity. A user with very regular habits will be easy to model, and

any intruder attempting to masquerade as such a user would likely exhibit behavior that deviated significantly from the user's normal activity. The actions of a system administrator, however, might be more irregular and harder to distinguish from the actions of an attacker. In addition, a hacker may be able to slowly change the characteristics that an anomaly detection system considers "normal" by deviating only slightly from normal behavior over a long period of time. After the anomaly detection system had been trained to consider more actions "normal" the attacker could mount an attack and avoid detection. A second disadvantage of anomaly detection schemes is the large computation and memory resources required to maintain the statistical model. Recent research on anomaly detection includes the development of EMERALD [46], which combines statistical anomaly detection from NIDES with signature verification.

Specification-based intrusion detection [39] is a second approach that can be used to detect new attacks. It detects attacks that make improper use of system or application programs. This approach involves first writing security specifications that describe the normal intended behavior of programs. Host-based audit records are then monitored to detect behavior that violates the security specifications. This approach was applied to 15 UNIX system programs and successfully found many attacks [39]. Specification-based intrusion detection has the potential to provide very low false alarm rates and detect a wide range of attacks including many forms of malicious code such as trojan horses, viruses, attacks that take advantage of race conditions, and attacks that take advantage of improperly synchronized distributed programs. Unfortunately, it is difficult to apply because security specifications must be written for all monitored programs. This is difficult because system and application programs are constantly updated. Specification-

based intrusion detection is thus best applied to a small number of critical user or system programs that might be considered prime targets for an attack.

The final strategy shown in Figure 2-1 is bottleneck verification. The bottleneck verification approach applies to situations where there are only a few, well defined ways to transition between two groups of states. One example of such a well-defined transition is transitions from a normal user to superuser within a shell. If an individual is in the user state, the only way to legally gain root privileges is by using the su command and entering the root password. Thus, if a bottleneck verification system can detect a shell being launched, determine the permissions of the new shell, and detect the successful use of the su command to gain root access (or, more importantly, the lack of a successful su command), then illegal transitions from normal user to root user can be detected—even if the transition is being made through some novel method that did not exist when the bottleneck verification system was created [65].

Chapter 3

Simulation Network

The goal of the 1998 DARPA Intrusion Detection System Evaluation was to collect and distribute the first standard corpus for evaluation of intrusion detection systems. This corpus was designed to evaluate both false alarm rates and detection rates of intrusion detection systems using many types of both known and novel attacks embedded in a large amount of normal background traffic. One roadblock that has discouraged the creation of such a corpus is the reluctance of companies and government agencies to release data collected from operational computer networks. Data collected from an operational computer network is optimal for the evaluation of intrusion detection systems, but this data may contain personal or sensitive information that could not be released to the many parties who conduct intrusion detection research. For this reason, all data in the 1998 DARPA Intrusion Detection System evaluation was synthesized and recorded on a network which *simulated* an operational network connected to the Internet [25][42].

3.1 Modeling an Air Force Local Area Network

The goal of the simulation network (or simnet) was to accurately simulate the network traffic of a Local Area Network that one might find at a United States Air Force facility. Automatically generated traffic used more than 20 network services, including dns, finger, ftp, http, ident, ping, pop, smtp, snmp, telnet, time, and X. In order to accurately

model the features of an Air Force network, statistics were measured from months of actual network traffic that was collected from more than fifty Air Force computer networks. Traffic statistics for automatically generated traffic matched average Air Force statistics. The content of email, ftp, web sites, and files was similar to actual documents. Content was generated using open-source documentation or statistical reconstruction from a large set of unclassified documents that preserved both unigram word frequencies and also the frequency of two-word sequences. Generating the data from public sources allowed the data to be distributed without security or privacy concerns.

The 32 different types of attacks that were inserted in the data collected on the simulation network represent attacks that one would expect to see on an Air Force network comprised mostly of UNIX workstations. These attacks were chosen to represent a mixture of different categories of computer attacks, and to provide a mix of older, more recent, and even some novel attacks. In some cases, transcripts of actual Air Force intrusions were used to develop attack scenarios. When such transcripts were not available, attacks were chosen from publicly known attacks and accounts of intrusions on civilian computer systems. Later sections of this thesis describe the attacks in detail.

3.2 Simulation Hardware and Network Topology

The simulation network consisted of two Ethernet network segments connected to each other through a router. A diagram of the network topology is shown in Figure 3-1. The router is located in the top center of this picture and is labeled "CISCO". Everything to the left of the router in the figure is considered the "outside" of the network and everything to the right of the router is the "inside" of the simulation network. The inside network is connected to one interface of the router and consists of all the computers that

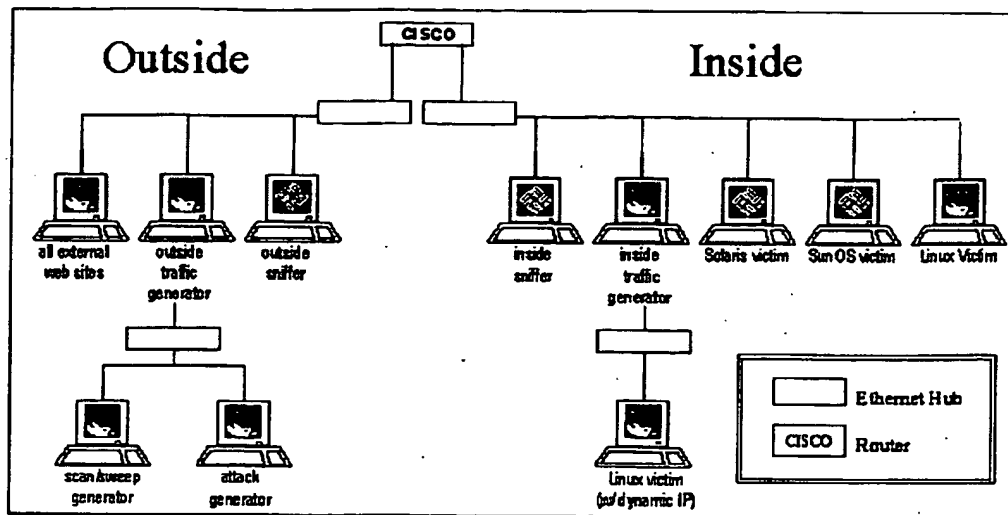


Figure 3-1: Simulation Network Topology

are part of the fabricated “eyrie.af.mil” domain. The computers that model the rest of the Internet are connected to the external interface of the router.

The simulation network included eleven computers. The outside of the network contains a traffic generator (for both background traffic and automated attacks), a web server, a sniffer/recorder, and two machines used for non-automated attack generation. The inside of the network (eyrie.af.mil) consisted of a background traffic generator, a sniffer/recorder, a Solaris 2.5 victim, a SunOS 4.1.4 victim, a Linux 4.2 victim and a Linux 5.0 victim that could dynamically change IP addresses. Although two computers (one inside, one outside) generated all of the background traffic in the simulation, a modification to the operating system of these computers enabled them to act as hundreds of “virtual” machines [26]. The same modification was made to the outside web server so this machine could mimic thousands of web servers. The modification to the operating system that created these virtual machines will be discussed in the next section.

3.3 Simulation Software

The hardware for the simulation network consists of only eleven machines and one router. Custom software allows this simple network to model the interaction of thousands of clients and servers. Without such software the simulation network would consist of hundreds of computers and the evaluation would be impractical in terms of both the cost and time required to maintain the network hardware.

3.3.1 Virtual Machines

The two traffic generators on the simulation network were configured with a modification to the Linux kernel provided by the Air Force Research Laboratory [26]. The IP swapping kernel allows each process that is started on the traffic generator to specify which IP address should be used as the source address for all network traffic generated by that process. At the time of invocation of a new process, an entry is added to a table (called the IP swap table) that maps process id to IP address. Whenever a network packet is generated, the kernel checks the process id of the process that generated the packet and uses the entry in the IP swap table as the source address of the new packet. The modified kernel was an important step in creating the illusion of a virtual machine, but there are several other modifications that support the creation of a single host that can act as many hosts. Daemons that provide network services—including telnetd, ftpd, and login—display banners that identify the machine they are running on. These daemons were modified to give different identification information depending on the destination IP address specified in the request.

3.3.2 Traffic Generation

Automatic traffic regeneration was used to create automated network sessions containing both artificial normal traffic and attacks that appear to be generated by humans working on a system. The traffic regeneration mechanism was designed to be:

- Automatic, requiring no human intervention.
- Reproducible; when repeated, sessions produce identical results.
- Robust, so it can run for long periods without human supervision.

As suggested in [48], the *expect* language was used to automate interaction between a system and a user, allowing autonomous sessions to be run as if a user were typing at a keyboard. Unfortunately, *expect* cannot be used to automate the interaction between a user and a graphical environment (such as X Windows), so any sessions that required interaction with a graphical user interface were run by human actors during the simulation. Because the amount of interaction included in the simulation was so large, creating a separate *expect* script for each session would have been unwieldy. For this reason, an *expect* program called the “regenerator” (for “traffic regenerator”) was written that reads information about a particular session from a specially formatted “exs” file. Each “exs” file contains a header and a body. Within the header is information about what time to start a session, the IP address of the local machine, the IP address of the remote machine, and a list of prompts we expect to see during this session. The body of the “exs” file contains one or more prompt-response pairs. For each prompt-response pair the regenerator first waits for the prompt indicated, pauses an indicated amount of time, and then types the command. Each session within the simulation is represented by a single “exs” script. To simulate a day of traffic, a day of “exs” scripts can be collected

and passed to the regenerator which then runs all of the sessions. If an error occurs in generating or collecting this traffic, the traffic collection can be repeated by rerunning the regenerator with the same "exs" scripts. The regenerator satisfies the need for automatic, reproducible, and robust traffic generation.

Chapter 4

Exploits

A large sample of actual computer attacks is needed to test an intrusion detection system. These attacks should cover the different classes of attack types and contain exploits for both newly discovered as well as older well-known vulnerabilities. An attack instance might consist of several phases. For example, an attacker might copy a program onto a system, run this program that exploits a system vulnerability to gain root privileges, and then use this root privilege to install a backdoor into the system for later access. This chapter provides some background on the different types of computer exploits, including a taxonomy that can be used to organize these exploits into meaningful groups.

4.1 Sources

Many of the exploits developed for the 1998 DARPA evaluation were drawn from ideas or implementations available from public sources on the Internet. Rootshell [50] is a web-site dedicated to collecting computer exploits and has a sizeable archive of attacks for many popular operating systems. The "Bugtraq" mailing list also frequently hosts "exploit code"—ostensibly released for the purpose of testing one's own system for vulnerability—when a new vulnerability is discussed. A searchable archive of the Bugtraq mailing list can be found on the Internet at <http://www.geek-girl.com>. Other exploits were created from information released by computer security groups such as

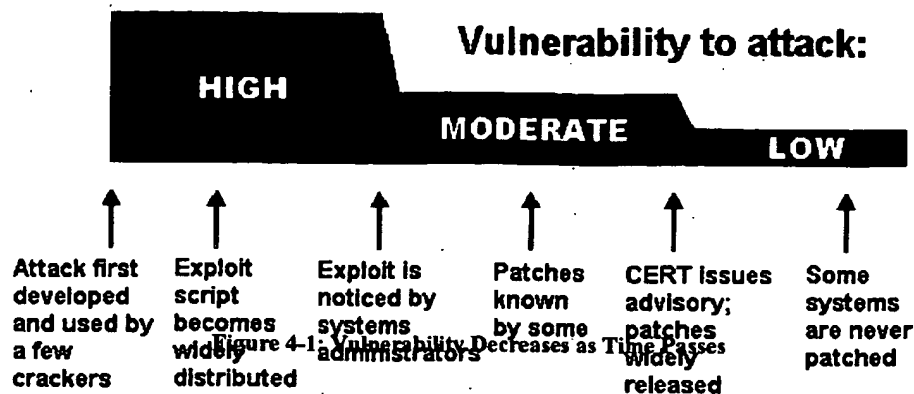


Figure 4-1: Vulnerability to a Newly Discovered Attack Decreases with Time

CERT [22] and ISS X-force [35] that frequently release information about new vulnerabilities. Additional sources of information about system vulnerabilities and possible exploits were vendor-initiated bulletins posted by operating system vendors like Sun Microsystems and Redhat Software. These bulletins are released to customers to encourage them to download patches that eliminate a new vulnerability.

New and novel exploits were also created specifically for the purpose of the evaluation. These new exploits are useful for determining how well an intrusion detection system works against novel attacks that were not publicly known at the time the intrusion detection system was developed.

4.2 Age of an Exploit

Each new exploit has a period of time during which it is most dangerous. Figure 4-1 is a simplified illustration of the various phases of a newly discovered exploit. The vertical axis of the figure shows vulnerability to attack and the horizontal axis represents time. As time progresses, more people are made aware of the vulnerability and patch their

systems to make them resistant to the exploit. Even after news of a particular vulnerability has become widespread, some systems might not be patched. Some computer systems with less-experienced administrators go years without having widely-known security holes fixed. Some of these older attacks were included in the set of attacks used for the 1998 DARPA evaluation. Intrusion detection systems were generally able to find these older, well-known attacks.

4.3 A Taxonomy for Computer Attacks

A taxonomy for classifying computer attacks was used to choose exploits for the evaluation. A good taxonomy makes it possible to classify attacks into groups that share common properties. Once these groups have been identified, the job of adequately testing an intrusion detection system becomes easier because instead of developing every possible attack we can choose a representative subset from each group. The taxonomy presented here was originally presented in [64]. The features of this taxonomy are:

- Each attack can be reliably placed in one category.
- All possible intrusions have a place in the taxonomy
- The taxonomy can be extended in the future.

This taxonomy was created for the express purpose of testing and evaluating intrusion detection systems. Within the taxonomy, each attack can be categorized as one of the following:

- A user performs some action at one level of privilege
- A user makes an unauthorized transition from a lower privilege level to a higher privilege level.

- A user stays at the same privilege level, but performs some action at a higher level of privilege.

The taxonomy requires a way of describing each privilege level, a way to describe transitions, and a way of categorizing actions. These three requirements are presented in the following sections.

4.3.1 Privilege Levels

The taxonomy defines an approach to ranking levels of privilege. The privilege categories that are applied within this thesis are:

R	Remote network access
L	Local network access
U	User access
S	Root/Super-user access
P	Physical Access to Host

Having privilege at the "Remote network access" level refers to having, via an interconnected network of systems, minimal network access to a target system. "Local network access" represents the ability to read from and write to the local network that the target machine uses. "User access" refers to the ability to run normal user commands on a system. "Root/Superuser access" gives a user total software control of a system. "Physical Access to Host" allows the operator to physically manipulate characteristics of the system (i.e. remove disk drives, insert floppy disks, turn the system off). This list only represents a subset of all possible access levels, but these were the most useful categories for describing the attacks in the 1998 DARPA intrusion detection evaluation.

4.3.2 Methods of Transition or Exploitation

An attacker needs to exploit some failure of a security framework in order to perform an attack. The five methods of transition that were explored for the 1998 DARPA evaluation and the single letters (m,a,b,c,s) used to represent the methods were:

- m) Masquerading:** In some cases it is possible to fool a system into giving access by misrepresenting oneself. Examples of masquerading include using a stolen username/password or sending a TCP packet with a forged source address.
- a) Abuse of Feature:** There are legitimate actions that one can perform, or is even expected to perform, that when taken to the extreme can lead to system failure. Example include filling up a disk partition with user files or starting hundreds of telnet connections to a host to fill its process table.
- b) Implementation Bug:** A bug in a trusted program might allow an attack to proceed. Specific examples include buffer overflows and race conditions.
- c) System Misconfiguration:** An attacker can exploit errors in security policy configuration that allows the attacker to operate at a higher level of privilege than intended.
- s) Social Engineering:** An attacker may be able to coerce a human operator of a computer system into giving the attacker access.

An individual attack may use more than one of these methods. For example, a bug in the implementation of the TCP stack on some systems makes it possible to crash the system by sending it a carefully constructed malformed TCP packet. This packet may also have

the source address forged so as to avoid identification of the attacker. Such an attack would be exploiting both masquerading and an implementation bug, and it would be possible to detect the intrusion by noting either of these features.

4.3.3 Transitions Between Privilege Levels

To show a transition between two privilege levels the strings for the two levels are written adjacent to one another with the method of transition between them. Two examples are shown in the following table:

Examples:

<i>Attack</i>	<i>String</i>	<i>Description</i>
Format	U-b-S	User exploits a bug in the format program to become root/superuser
Ftp-write	R-c-U	A user with remote network access exploits a badly configured anonymous ftp server to gain local user access

4.3.4 Actions

There are many actions that can occur as part of a computer attack. Within the taxonomy, actions are represented with a string that represents a category, and a specification string that describes the specific action taken. For example, the string **Probe(Users)**, represents some action taken by an attacker to gather information about the users of a system. The following paragraphs describe the five categories of actions that were used to describe the actions taken during the 1998 DARPA intrusion detection evaluation.

<i>Category</i>	<i>Specific Type</i>	<i>Description</i>
Probe	Probe(Machines)	Determine types and numbers of machines on a network
	Probe(Services)	Determine the services a particular system supports
	Probe(Users)	Determine the names or other information about users with accounts on a given system
Deny	Deny(Temporary)	Temporary Denial of Service with automatic recovery
	Deny(Administrative)	Denial of Service requiring administrative intervention
	Deny(Permanent)	Permanent alteration of a system such that a particular service is no longer available
Intercept	Intercept(Files)	Intercept files on a system
	Intercept(Network)	Intercept traffic on a network
	Intercept(Keystrokes)	Intercept keystrokes pressed by a user
Alter	Alter(Data)	Alteration of stored data
	Alter(Intrusion-Traces)	Removal of hint of an intrusion, such as entries in log files
Use	Use(Recreational)	Use of the system for enjoyment, such as playing games or bragging on IRC
	Use(Intrusion-Related)	Use of the system as a staging area/entry point for future attacks

Figure 4-2: Summary of Possible Types of Actions.

Probes are actions taken by an attacker to gather information about one or more machine. Probes are represented within the taxonomy by the category label of “**Probe**”. The specific types of probes used in the DARPA evaluation were: (1) Probing a network to see how many and what types of machines are on that network (**Probe(Machines)**), (2) Probing a system to see what services the system supports (**Probe(Services)**), and (3) Probing a system to find out information about user accounts on that system (**Probe(Users)**).

Denial of service attacks are attempts to interrupt or degrade a service that a system provides. These attacks are represented within the taxonomy by the category label **"Deny"**. The classes of denial of service attacks used in the DARPA evaluation were: (1) Temporary denial of service with automatic recovery (**Deny(Temporary)**), (2) Denial of service requiring administrative action for recovery (**Deny(Administrative)**), and (3) Permanent denial of service with total system reconstruction required for recovery (**Deny(Permanent)**).

Another category of attacker actions is the interception of data. Interception of data is represented within the taxonomy by the category label **"Intercept"**. The types of data interception actions used in the 1998 DARPA evaluation were: (1) Interception/Reading of files on a file system (**Intercept(Files)**), and (2) Interception of packets on a network (**Intercept(Network)**).

An additional category of action is the alteration or creation of data on a system or network. Actions that involve data alteration or creation are represented with the category label **"Alter"**. The types of data alteration used in the 1998 DARPA evaluation were: (1) Alteration of data stored on a system, such as a password file or any other file (**Alter(Data)**), and (2) Removal of hints of an intrusion, such as entries in log files (**Alter(Intrusion-Traces)**).

The final category of attacker action described in the taxonomy is "use" of a system. Any use of the system that does not fall into the categories described above can be placed in the category represented by the category label **"Use"**. The specific ways in which an attacker might use a system that were included in the 1998 DARPA evaluation were: (1) Use of the system by the intruder for enjoyment or recreational purposes such

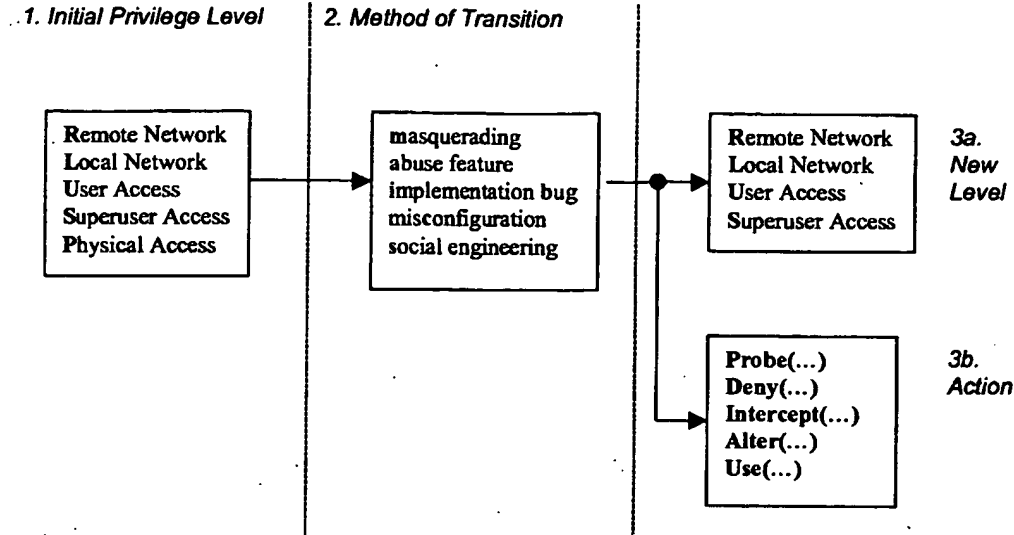


Figure 4-3: A Summary of Possible Attack Descriptions

as playing games or bragging on IRC (**Use(Recreational)**), and (2) Use of a system as a staging ground or entry point for attacks on other systems (**(Use(Intrusion-Related))**).

The action categories and specifications described in the previous paragraphs are summarized in Table 4-2. Each row of this table represents a specific type of action within a category. These specific actions have been grouped according to the categories presented above. The first column of the table is the action category, the second column is the string that represents the action in the taxonomy, and the third column in each row is a description of that particular type of action.

4.3.5 Using the Taxonomy to Describe Attacks

Figure 4-3 gives an overview of the approach the taxonomy uses for the classification of attacks. Each attack classified by this taxonomy is represented as a short alpha-numeric string. The sub-strings in this description are the bold strings from the previous three

sections. The initial privilege level is indicated by **R**, **L**, **U**, **S**, or **P** as defined in section 4.3.1, the actions are indicated by any of the strings presented in section 4.3.4, and the method of exploitation is indicated by **m**, **a**, **b**, **c**, or **s** as defined in section 4.3.2.

In order to describe an attack, select the privilege level that the attacker had before the attack occurred, from the possible choices of Remote Network, Local Network, Local User, Superuser/Root, and Physical Access. Next, select the method of exploitation, if the method is known. If the method is unknown, then a question mark ("?",) is used to indicate the method of exploitation. The possible choices are masquerading, abuse of feature, implementation bug, misconfiguration, or social engineering. Finally, either indicate the level of privilege the attacker gained as a result of the exploit (again with **R**, **L**, **U**, **S**, or **P**) or the actions the user performed at the current level of privilege. The taxonomy presented in [64] has additional features that are not discussed here, but the material presented in the section provides a subset of the taxonomy that is useful for the purpose of discussing the attacks in the 1998 DARPA intrusion detection evaluation.

4.3.6 Examples

The following table presents three examples that show the correct formatting of the alphanumeric string that specifies an action being performed at a specific privilege level. In a SYN flood (or neptune) attack the attacker sends a stream of SYN packets to a port on a target machine. For a short period of time after these packets have been sent, other users are unable to access the network services provided by that port. In the second example a user runs the crack program to decrypt the password file of a machine that has been compromised. In the third example, the attacker uses the Ffbconfig attack to make a

transition from Local User privilege to Root/Superuser privilege, and then uses this new privilege level to alter the password file on the victim system.

Examples:

<i>Attack</i>	<i>String</i>	<i>Description</i>
SYN flood	R-a-Deny(temporary)	A user with remote network access temporarily denies service
Cracking passwords	U-Use(Intrusion)	A user with a local account runs a program which attempts to decrypt entries in the password file.
ffbconfig	U-b-S-Alter(Files)	An attacker with a local account uses a bug in the Ffbconfig program to gain root access and alter files.

Chapter 5

Exploits for the 1998 DARPA Evaluation

Figure 5-1 shows the 32 different exploits that were used in the 1998 DARPA intrusion detection evaluation. This table presents the attacks broken up into categories of type and

Figure 5-1: The Attacks Used in the 1998 DARPA Intrusion Detection Evaluation

	Solaris	SunOS	Linux
Denial Of Service (R-Deny)	Apache2 Back Mailbomb Neptune Ping Of Death Process Table Smurf Syslogd UDP Storm	Apache2 Back Land Mailbomb Neptune Ping of death Process Table Smurf UDP Storm	Apache2 back Mailbomb Neptune Ping of death Process Table Smurf Teardrop UDP Storm
Remote to User (R-?-U)	dictionary ftp-write guest phf xlock xsnoop	dictionary ftp-write guest phf xlock xsnoop	dictionary ftp-write guest imap named phf sendmail xlock xsnoop
User to Super-user (U-?-S)	eject ffbconfig fdformat ps	loadmodule ps	perl xterm
Surveillance/ Probing (R-Probe)	ip sweep mscan nmap saint satan	ip sweep mscan nmap saint satan	ip sweep mscan nmap saint satan

vulnerable operating system. The four type categories represent groupings of the possible attack types listed in the taxonomy. These four groups are: Denial of Service (R-?-Deny), Remote to Local User (R-?-U), Local User to Super-user (U-?-S), and Probes (R-?-Probe). The three columns of the table divide the exploits by target platform. Some attacks are listed in more than one column. The Smurf attack, for example, is listed three times—in the Solaris column, the SunOS column, and the Linux column—because all three operating systems are vulnerable to the Smurf attack. The next four chapters present detailed descriptions of each class of attack, and the individual attacks from that class that were included in the 1998 DARPA intrusion detection evaluation

Chapter 6

Denial of Service Attacks

A denial of service attack is an attack in which the attacker makes some computing or memory resource too busy or too full to handle legitimate requests, or denies legitimate users access to a machine. There are many varieties of denial of service (or DoS) attacks. Some DoS attacks (like a mailbomb, neptune, or smurf attack) abuse a perfectly legitimate feature. Others (teardrop, Ping of Death) create malformed packets that confuse the TCP/IP stack of the machine that is trying to reconstruct the packet. Still others (apache2, back, syslogd) take advantage of bugs in a particular network daemon. Figure 6-1 provides an overview of the denial of service attacks used in the 1998 DARPA intrusion detection evaluation. Each row represents a single type of attack. The six columns show the attack name, a list of the services that the attack exploits, the platforms that are vulnerable to the attack, the type of mechanism that is exploited by the attack (implementation bug, abuse of feature, masquerading, or misconfiguration), a generalization of the amount of time the attack took to implement, and a summary of the effect of the attack. The following sections describe in detail each of the Denial of Service attacks that were included in the 1998 DARPA intrusion detection evaluation.

Name	Service	Vulnerable Platforms	Mechanism	Time to Implement	Effect
Apache2	http	Any Apache	Abuse	Short	Crash httpd
Back	http	Any Apache	Abuse/Bug	Short	Slow server response
Land	N/A	SunOS	Bug	Short	Freeze machine
Mailbomb	smtp	All	Abuse	Short	Annoyance
SYN Flood	Any TCP	All	Abuse	Short	Deny service on one or more ports for minutes
Ping of Death	icmp	None	Bug	Short	None
Process Table	Any TCP	All	Abuse	Moderate	Deny new processes
Smurf	icmp	All	Abuse	Moderate/Long	Network Slowdown
Syslogd	syslog	Solaris	Bug	Short	Kill Syslogd
Teardrop	N/A	Linux	Bug	Short	Reboot machine
Udpstorm	echo/ chargen	All	Abuse	Short	Network Slowdown

Figure 6-1: Summary of Denial of Service Attacks

6.1 Apache2 R-a-Deny(Temporary/Administrative)

Description

The Apache2 attack is a denial of service attack against an apache web server where a client sends a request with many http headers. If the server receives many of these requests it will slow down, and may eventually crash [4].

Simulation Details

This exploit was adapted from C code originally posted to the bugtraq mailing list. A C-shell wrapper was also created which executes the apache2 C program in a loop until the server being attacked is no longer responsive.

As soon as the attack was launched the load average (as reported by the "top" program) of the victim server jumped to 5 or more. As more and more requests were submitted to the web server the memory usage and load average of the victim continued to climb until eventually the httpd daemon ran out of memory and crashed. At this point the server no longer responded to http requests and the httpd daemon needed to be restarted by the superuser for service to be restored.

Attack Signature

Every http request submitted as part of this exploit contains many http headers. Although the exact number and value of these headers could be varied by an attacker, the particular version of the exploit which was used in the 1998 DARPA evaluation sent http GET requests with the header "User-Agent: sioux\r\n" repeated 10000 times in each request. The actual content of the header is not important for the exploit—the exploit is only dependent on the fact that http request contains *many* headers. A typical http request contains twenty or fewer headers, so the 10000 headers used by this exploit are quite anomalous.

6.2 Back

R-a-Deny(Temporary)

Description

In this denial of service attack against the Apache web server, an attacker submits requests with URL's containing many frontslashes. As the server tries to process these requests it will slow down and becomes unable to process other requests [55].

Simulation Details

The Back attack was implemented as a C shell script that used the Netcat [31] tool to generate network traffic. This shell script was adapted from a script originally posted to the Bugtraq mailing list. Although the number of frontslashes in the URL sent by the shell script could be varied, the number of frontslashes that was determined to be optimal for denial of service against Apache running on Linux 4.2 was between six and seven thousand.

The Back attack causes instances of the httpd process on the victim to consume excessive CPU time. This consumption of the CPU slows down all the system's activities, including responses to network requests. The system recovers automatically when the attack stops.

Attack Signature

An intrusion detection system looking for the Back attack needs to know that requests for documents with more than some number of frontslashes in the URL should be considered an attack. Certainly, a request with 100 frontslashes in the URL would be highly irregular on most systems. This threshold could be varied to find the desired balance between detection rate and false alarm rate.

6.3 Land

R-b-Deny(Administrative)

Description

The Land attack is a denial of service attack that is effective against some older TCP/IP implementations. The only vulnerable platform used in the 1998 DARPA evaluation was SunOS 4.1. The Land attack occurs when an attacker sends a spoofed SYN packet in which the source address is the same as the destination address [17].

Simulation Details

The land exploit program used in the DARPA evaluation was adapted from a C implementation found at <http://www.rootshell.com>. The exploit is quite simple and the code could easily be rewritten in any language with access to the TCP sockets interface. The code sends a single SYN packet with the source address spoofed to be the same as the destination address.

Within the simulation, this exploit was run against a Sun SPARC WorkStation running SunOS version 4.1. When a TCP SYN packet with an identical source and destination address was received by this host, the system completely locked up. In order to restore service, the machine had to be physically turned off and on again.

Attack Signature

The Land attack is recognizable because IP packets with identical source and destination addresses should never exist on a properly working network.

6.4 Mailbomb

R-a-Deny(Administrative)

Description

A Mailbomb is an attack in which the attacker sends many messages to a server, overflowing that server's mail queue and possibly causing system failure.

Simulation Details

This exploit was implemented as a perl program that constructed mail messages and connected to the SMTP port of the victim machine directly. The Mailbomb perl program accepted as parameters the e-mail addresses of victims as well as the number of e-mail messages to send.

Although the Mailbomb exploit was used several times throughout the simulation with different parameters, a typical attack would send 10,000 one kilobyte messages (10 megabytes of total data) to a single user. This volume of messages was not enough to adversely effect the performance of the server or cause system failure. As implemented, this attack was more of a nuisance for a particular user than a real threat to the overall security of a server.

Attack Signature

An intrusion detection system that is looking for a mailbomb attack can look for thousands of mail messages coming from or sent to a particular user within a short period of time. This identification is a somewhat subjective process. Each site might have a different definition of how many e-mail messages can be sent by one user or to one user before the messages are considered to be part of a mailbomb.

6.5 SYN Flood (Neptune)

R-a-Deny(Temporary)

Description

A SYN Flood is a denial of service attack to which every TCP/IP implementation is vulnerable (to some degree). Each half-open TCP connection made to a machine causes the "tcpd" server to add a record to the data structure that stores information describing all pending connections. This data structure is of finite size, and it can be made to overflow by intentionally creating too many partially-open connections. The half-open connections data structure on the victim server system will eventually fill and the system will be unable to accept any new incoming connections until the table is emptied out. Normally there is a timeout associated with a pending connection, so the half-open connections will eventually expire and the victim server system will recover. However, the attacking system can simply continue sending IP-spoofed packets requesting new connections faster than the victim system can expire the pending connections. In some cases, the system may exhaust memory, crash, or be rendered otherwise inoperative [13].

Simulation Details

The neptune exploit code used in the simulation was compiled from C code originally posted to the bugtraq archive. The neptune program allows the user to specify a victim host, the source address to use in the spoofed packets, the number of packets to send, and the ports to hit on the victim machine (including an "infinity" option that would attack all ports).

The neptune exploit was effective against all three of the victim machines used in the simulation. Every TCP/IP implementation is vulnerable to this attack to a varying degree depending on the size of the data structure used to store incoming connections and the

timeout value associated with half-open connections. As a point of reference, sending twenty SYN packets to a port on a Solaris 2.6 system will cause that port to drop incoming requests for approximately ten minutes. During the simulation, a neptune attack which sent 20 SYN packets to every port from 1 to 1024 of the Solaris server once every ten minutes was able block incoming connections to any of these ports for more than an hour.

Attack Signature

A neptune attack can be distinguished from normal network traffic by looking for a number of simultaneous SYN packets destined for a particular machine that are coming from an unreachable host. A host-based intrusion detection system can monitor the size of the tcpd connection data structure and alert a user if this data structure nears its size limit.

6.6 Ping Of Death

R-b-Deny(Temporary)

Description

The Ping of Death is a denial of service attack that affects many older operating systems. Although the adverse effects of a Ping of Death could not be duplicated on any victim systems used in the 1998 DARPA evaluation, it has been widely reported that some systems will react in an unpredictable fashion when receiving oversized IP packets. Possible reactions include crashing, freezing, and rebooting [14].

Simulation Details

Several implementations of the Ping of Death exploit can be found at <http://www.rootshell.com> as well as many other sources on the web. This exploit is popular because early versions of the ping program distributed with Microsoft Windows95 would allow the user to create oversize ping packets simply by specifying a parameter at the command line (i.e. ping -l 65510). Thus, many users could potentially exploit this bug without even making the effort to download and compile a program.

The Ping of Death attack affected none of the victim systems used in the evaluation. The attack was included as an example of an attempted known attack that fails to have an effect.

Attack Signature

An attempted Ping of Death can be identified by noting the size of all ICMP packets and flagging those that are longer than 64000 bytes.

6.7 Process Table

R-a-Deny(Temporary)

Description

The Process Table attack is a novel denial-of-service attack that was specifically created for this evaluation. The Process Table attack can be waged against numerous network services on a variety of different UNIX systems. The attack is launched against network services which fork() or otherwise allocate a new process for each incoming TCP/IP connection. Although the standard UNIX operating system places limits on the number of processes that any one user may launch, there are no limits on the number of processes that the superuser can create, other than the hard limits imposed by the operating system. Since incoming TCP/IP connections are usually handled by servers that run as root, it is possible to completely fill a target machine's process table with multiple instantiations of network servers. Properly executed, this attack prevents any other command from being executed on the target machine.

An example of a service that is vulnerable to this attack is the finger service. On most computers, finger is launched by inetd. The authors of inetd placed several checks into the program's source code that must be bypassed in order to initiate a successful process attack. In a typical implementation (specifics will vary depending on the actual UNIX version used), if inetd receives more than 40 connections to a particular service within 1 minute, that service is disabled for 10 minutes. The purpose of these checks was not to protect the server against a process table attack, but to protect the server against buggy code that might create many connections in rapid-fire sequence. To launch a successful process table attack against a computer running inetd and finger, the following sequence may be followed: 1. Open a connection to the target's finger port. 2. Wait for 4 seconds.

3. Repeat steps 1-2. This attack has been attempted against a variety of network services on a variety of operating systems. It is believed that the imap and sendmail servers are vulnerable. Most imap server software contains no checks for rapid-fire connections. Thus, it is possible to shut down a computer by opening multiple connections to the imap server in rapid succession. With sendmail the situation is reversed. Normally, sendmail will not accept connections after the system load has jumped above a predefined level. Thus, to initiate a successful sendmail attack it is necessary to open the connections very slowly, so that the process table keeps growing in size while the system load remains more or less constant [6].

Simulation Details

The version of this exploit used in the simulation was implemented as a perl script that would open connections to a port every *n* seconds, where the port and the number of seconds *n* are specified as run-time parameters. The connections were maintained until a user terminated the script.

The number of connections that must be opened before denial of service is accomplished depends on the size of the process table in the operating system of the victim machine. By using the Process Table attack on the fingerd port as described above, the process table of a Solaris server was exhausted after opening approximately 200 connections (at a rate of one connection every four seconds it took about 14 minutes before the process table was full). After the process table was full, no new process could be launched on the victim machine until the attack was terminated by the attacking program or an administrator manually killed the connections initiated by the attacking script (which is quite difficult to do without launching a new process).

Attack Signature

Because this attack consists of abuse of a perfectly legal action, an intrusion detection system that is trying to detect a process table attack will need to use somewhat subjective criteria for identifying the attack. The only clue that such an attack is occurring is an "unusually" large number of connections active on a particular port. Unfortunately "unusual" is different for every host, but for most machines, hundreds of connections to the finger port would certainly constitute unusual behavior.

6.8 Smurf

R-a-Deny(Temporary)

Description

In the "smurf" attack, attackers use ICMP echo request packets directed to IP broadcast addresses from remote locations to create a denial-of-service attack. There are three parties in these attacks: the attacker, the intermediary, and the victim (note that the intermediary can also be a victim) [18]. The attacker sends ICMP "echo request" packets to the broadcast address (xxx.xxx.xxx.255) of many subnets with the source address spoofed to be that of the intended victim. Any machines that are listening on these subnets will respond by sending ICMP "echo reply" packets to the victim. The smurf attack is effective because the attacker is able to use broadcast addresses to amplify what would otherwise be a rather innocuous ping flood. In the best case (from an attacker's point of view), the attacker can flood a victim with a volume of packets 255 times as great in magnitude as the attacker would be able to achieve without such amplification. This amplification effect is illustrated by Figure 6-2. The attacking machine (located on the left of the figure) sends a single spoofed packet to the broadcast address of some network, and every machine that is located on that network responds by sending a packet to the victim machine. Because there can be as many as 255 machines on an ethernet segment, the attacker can use this amplification to generate a flood of ping packets 255 times as great in size (in the best case) as would otherwise be possible. This figure is a simplification of the smurf attack. In an actual attack, the attacker sends a *stream* of icmp "ECHO" requests to the broadcast address of *many* subnets, resulting in a large, continuous stream of "ECHO" replies that flood the victim.

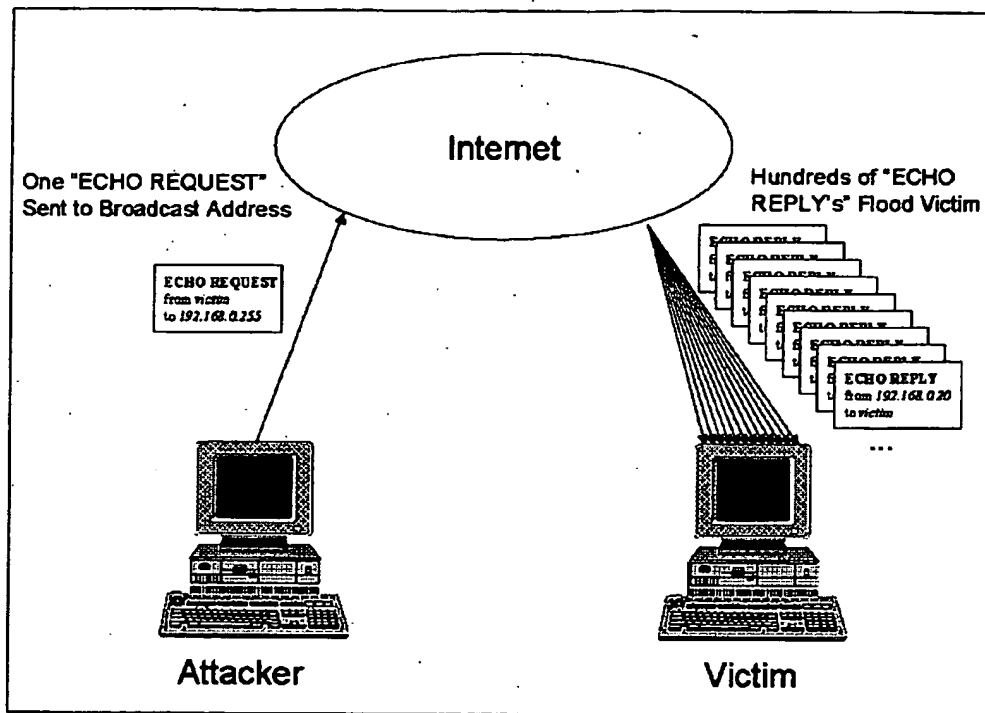


Figure 6-2: The Internet is Used to Amplify a Ping Flood and Create a Smurf Attack

Simulation Details

Because the simulation network for the 1998 DARPA evaluation has a flat network topology with only two physical subnets, the smurf attack as described above could not be implemented on the simulation network. For this reason, the “smurfsim” program was developed to recreate the observable effects of a smurf attack. Smurfsim uses the raw socket API to construct ICMP packets with forged source addresses. Smurfsim takes as parameters the IP address of the victim, the number of packets to send, the average percentage of hosts on a subnet that are alive, and a comma-separated list of subnets. The program then randomly constructs a list of hosts that are alive on each of the subnets in the comma-separated list and starts sending “echo reply” packets to the victim, that have been spoofed to look like they originating from the hosts in the list. This behavior is

exactly what would occur if an attacker had performed an actual Smurf attack in which "echo request" packets (with the source address spoofed to be that of the victim machine) were sent to the broadcast address of each subnet given in the parameter list.

Several different simulated Smurf attacks were included in the evaluation data. In the most extreme case, the smurfsim program was used to simulate a smurf attack that generated traffic from 100 subnets for a period of one hour. During this period of time the entire simulation network was unresponsive and other network sessions (such as normal users trying to send e-mail, etc) would time out before they could be completed. In all, this particular attack instance generated over two gigabytes of network packets.

Attack Signature

The Smurf attack can be identified by an intrusion detection system that notices that there are a large number of "echo replies" being sent to a particular victim machine from many different places, but no "echo requests" originating from the victim machine.

6.9 Syslogd

R-b-Deny(Administrative)

Description

The Syslogd exploit is a denial of service attack that allows an attacker to remotely kill the syslogd service on a Solaris server. When Solaris syslogd receives an external message it attempts to do a DNS lookup on the source IP address. If this IP address doesn't match a valid DNS record, then syslogd will crash with a Segmentation Fault [54].

Simulation Details

The Syslogd exploit used in the 1998 DARPA evaluation was a C program that was originally posted to the Bugtraq mailing list. This code was compiled to create an exploit program that was used to remotely cause the syslogd program to crash on a Solaris 2.5 server. Once syslogd has crashed it must be manually restarted by an administrator for the logging service to be restored.

Attack Signature

The one way to reliably recognize this attack with a network-monitoring intrusion detection system is to notice a packet destined for the syslog port that contains an unreachable source address. Of course, it may not be realistic for an intrusion detection system to check every packet destined for the syslog port to see whether or not the source address is resolvable. If no remote system logging is expected to occur on a particular network, any external syslog messages appearing on this network is likely to be an attack. Finally, a host-based intrusion detection system could be configured to notice the syslog process die because of a segmentation fault.

6.10 Teardrop

R-a-Deny(Temporary)

Description

The teardrop exploit is a denial of service attack that exploits a flaw in the implementation of older TCP/IP stacks. Some implementations of the IP fragmentation re-assembly code on these platforms does not properly handle overlapping IP fragments [17].

Simulation Details

The teardrop name is derived from a widely available C program that exploits this vulnerability. This exploit code can be found at <http://www.rootshell.com> and in the Bugtraq archives. Although many systems are rumored to be vulnerable to the teardrop attack, of the systems used in the DARPA evaluation, only the Redhat Linux 4.2 systems were vulnerable. The teardrop attack would cause these machines to reboot.

Attack Signature

An intrusion detection system can find this attack by looking for two specially fragmented IP datagrams. The first datagram is a 0 offset fragment with a payload of size N, with the MF bit on (the data content of the packet is irrelevant). The second datagram is the last fragment (MF = 0), with a positive offset greater than N and with a payload of size less than N [5].

6.11 Udpstorm

R-a-Deny(Administrative)

Description

A Udpstorm attack is a denial of service attack that causes network congestion and slowdown. When a connection is established between two UDP services, each of which produces output, these two services can produce a very high number of packets that can lead to a denial of service on the machine(s) where the services are offered. Anyone with network connectivity can launch an attack; no account access is needed. For example, by connecting a host's chargen service to the echo service on the same or another machine, all affected machines may be effectively taken out of service because of the excessively high number of packets produced. An illustration of such an attack is presented in Figure 6-2. The figure demonstrates how an attacker is able to create a never-ending stream of packets between the echo ports of two victims by sending a single spoofed packet. First, the attacker forges a single packet that has been spoofed to look like it is coming from the echo port on the first victim machine and sends it to the second victim. The echo service blindly responds to any request it receives by simply echoing the data of the request back to the machine and port that sent the echo request, so when the victim receives this spoofed packet it sends a response to the echo port of the second victim. This second victim responds in like kind, and the loop of traffic continues until it is stopped by intervention from an external source [10].

Simulation Details

Code that exploits this vulnerability was posted to the bugtraq mailing list. This program sends a single spoofed UDP packet to a host. This single spoofed packet is able to create a never-ending stream of data being sent from the echo port of one machine to the echo

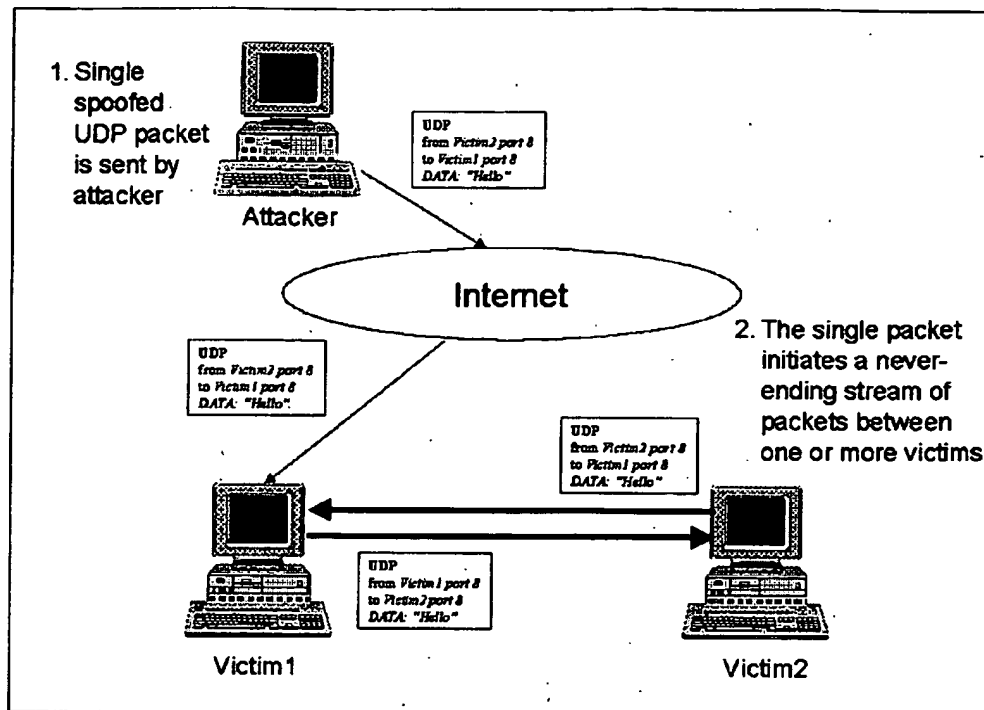


Figure 6-3: UDPStorm is Triggered by a Single Spoofed Packet

port of another. This loop created network congestion and slowdown that would continue until the inetd daemon was restarted on one of two victim machines.

Attack Signature

This attack can be identified in two ways. First, the single packet that initiates the attack can be recognized because it is a packet originating from outside the network that has been spoofed to appear as if it is coming from a machine inside the network. Second, once the loop of network traffic has been initiated, an intrusion detection system that can see network traffic on the inside of the network can note that traffic is being sent from the chargen or echo port of one machine to the chargen or echo port of another.

Chapter 7

User to Root Attacks

User to Root exploits are a class of exploit in which the attacker starts out with access to a normal user account on the system (perhaps gained by sniffing passwords, a dictionary attack, or social engineering) and is able to exploit some vulnerability to gain root access to the system.

There are several different types of User to Root attacks. The most common is the buffer overflow attack. Buffer overflows occur when a program copies too much data into a static buffer without checking to make sure that the data will fit. For example, if a program expects the user to input the user's first name, the programmer must decide how many characters that first name buffer will require. Assume the program allocates 20 characters for the first name buffer. Now, suppose the user's first name has 35 characters. The last 15 characters will overflow the name buffer. When this overflow occurs, the last 15 characters are placed on the stack, overwriting the next set of instructions that was to be executed. By carefully manipulating the data that overflows onto the stack, an attacker can cause arbitrary commands to be executed by the operating system. Despite the fact that programmers can eliminate this problem through careful programming techniques, some common utilities are susceptible to buffer overflow attacks [2]. Another class of User to Root attack exploits programs that make assumptions about the environment in which they are running. A good example of such an attack is the

Name	Service	Vulnerable Platforms	Mechanism	Time to Implement	Effect
Eject	Any user session	Solaris	Buffer Overflow	Medium	Root Shell
Ffbconfig	Any user session	Solaris	Buffer Overflow	Medium	Root Shell
Fdformat	Any user session	Solaris	Buffer Overflow	Medium	Root Shell
Loadmodule	Any user session	SunOS	Poor Environment Sanitation	Short	Root Shell
Perl	Any user session	Linux	Poor Environment Sanitation	Short	Root Shell
Ps	Any user session	Solaris	Poor Temp File Management	Short	Root Shell
Xterm	Any user session	Linux	Buffer Overflow	Short	Root Shell

Figure 7-1: Summary of User to Root Attacks

loadmodule attack, which is discussed below. Other User to Root attacks take advantage of programs that are not careful about the way they manage temporary files. Finally, some User to Root vulnerabilities exist because of an exploitable race condition in the actions of a single program, or two or more programs running simultaneously [27]. Although careful programming could eliminate all of these vulnerabilities, bugs like these are present in every major version of UNIX and Microsoft Windows available today.

Figure 7-1 summarizes the User to Root attacks used in the 1998 DARPA evaluation. Note from this table that all of the User to Root attacks can be run from any interactive user session (such as by sitting at the console, or interacting through telnet or rlogin), and that all of the attacks spawn a new shell with root privileges. The following sections describe each of the User to Root attacks that was used in the 1998 DARPA intrusion detection evaluation in greater detail.

7.1 Eject

U-b-S

Description

The Eject attack exploits a buffer overflow in the "eject" binary distributed with Solaris 2.5. In Solaris 2.5, removable media devices that do not have an eject button or removable media devices that are managed by Volume Management use the eject program. Due to insufficient bounds checking on arguments in the volume management library, libvolmgt.so.1, it is possible to overwrite the internal stack space of the eject program. If exploited, this vulnerability can be used to gain root access on attacked systems[60].

Simulation Details

A truncated version of the eject exploit that was used in the 1998 evaluation is shown in Figure 7-2. This exploit was originally posted to the bugtraq mailing list. The exploit script, once compiled, can be run in a command line session on a Solaris server to spawn a shell that ran with root privileges.

Attack Signature

There are several ways that an intrusion detection system might identify this attack. Assuming an attacker already has access to an account on the victim machine and is running the exploit as part of a remote session, a network-based system can look at the contents of the telnet or rlogin session the attacker is using and notice one of several features.

First, assuming that an attacker transmits the C code to the victim machine unencrypted, the intrusion detection system could look for specific features of the source code. For example, an intrusion detection system could look for the string "Jumping to

```

1 #include <stdio.h>
2 #include <stdlib.h>
3 #include <sys/types.h>
4 #include <unistd.h>
5
6 #define BUF_LENGTH 264
7 #define EXTRA 36
8 #define STACK_OFFSET 8
9 #define SPARC_NOP 0xc013a61c
10 u_char sparc_shellcode[] =""
...
17 ";
18
19 u_long get_sp(void)
20 {
21     __asm__ ("mov %sp,%i0 \n");
22 }
23
24 void main(int argc, char *argv[])
25 {
26     char buff[BUF_LENGTH + EXTRA + 8];
27     long targ_addr;
28     u_long *long_p;
29     u_char *char_p;
30     int i, code_length = strlen(sparc_shellcode), dso=0;
31
32     if(argc > 1) dso=atoi(argv[1]);
33
34     long_p=(u_long *) buff;
35     targ_addr = get_sp() - STACK_OFFSET - dso;
36
37     for (i = 0; i < (BUF_LENGTH - code_length) / sizeof(u_long); i++)
38         *long_p++ = SPARC_NOP;
...
43     for (i = 0; i < EXTRA / sizeof(u_long); i++)
44         *long_p++ = targ_addr;
45     printf("Jumping to address 0x%x B[%d] E[%d] SO[%d]\n",
46         targ_addr, BUF_LENGTH, EXTRA, STACK_OFFSET);
47     execl("/bin/eject", "eject", & buff, (char *) 0);
48     perror("execl failed");
49 }

```

*Unique Strings that can
be Used to Identify an
Eject Attack*

Figure 7-2: C code for the Eject Exploit

address" on line 45 of the source code or the line "execl ('/bin/eject' ; 'eject', & buff(char *) 0);" from line 47.

Even if the attacker encrypts the source code, the attack leaves a distinct signature. A segment of the transcript from an actual instantiation of this attack that was used in the simulation is shown below:


```
pascal> /tmp/162562
Jumping to address 0xeffff6a0 B[364] E[400] SO[704]
#
```

An intrusion detection system that saw only these three lines has several clues that an attack has taken place. First, the user's prompt has changed from "pascal>" to "#" without running the su command. Second, the string "Jumping to address" is again printed. Of course, a careful attacker would remove this line from the source code, but simply looking for the string "Jumping to address" would catch the less careful attacker. Finally, a host-based intrusion detection system could catch an eject attack either by noticing the invocation of the eject program with a large argument, or by performing bottleneck verification [43] on the transition from normal user to root user and noticing that the user did not make a legal user to root transition.

7.2 Ffbconfig

U-b-S

Description

The Ffbconfig attack exploits a buffer overflow in the "fbconfig" program distributed with Solaris 2.5. The fbconfig program configures the Creator Fast Frame Buffer (FFB) Graphics Accelerator, which is part of the FFB Configuration Software Package, SUNWfbcf. This software is used when the FFB Graphics accelerator card is installed. Due to insufficient bounds checking on arguments, it is possible to overwrite the internal stack space of the fbconfig program [61].

Simulation Details

This attack is very similar to the eject attack described above. Once again, C code to exploit this vulnerability was posted on the Bugtraq mailing list.

Attack Signature

The means by which an intrusion detection system can identify the fbconfig attack are similar to those described above for the Eject exploit. An attacker who is exploiting this vulnerability must first transfer the code for the exploit (either C code to be compiled, or pre-compiled code) onto the victim machine, and then run the exploit. As with the eject exploit, there are strings within the source code of the fbconfig exploit script which identify the attack to a network or host based intrusion detection system. A host-based intrusion detection system can perform bottleneck verification or look for the invocation of the command "/usr/sbin/fbconfig" with an oversized argument for the "-dev" parameter.

7.3 Fdformat

U-b-S

Description

The Fdformat attack exploits a buffer overflow in the "fdformat" program distributed with Solaris 2.5. The fdformat program formats diskettes and PCMCIA memory cards. The program also uses the same volume management library, libvolmgt.so.1, and is exposed to the same vulnerability as the eject program [60].

Simulation Details

Exploit code for this vulnerability was posted to the Rootshell Website [53] in March, 1997. The exploit code was used unmodified for the DARPA evaluation.

Attack Signature

Methods for identifying this attack are nearly identical to those described for the eject and ffbconfig attacks.

7.4 Loadmodule

U-b-S

Description

The Loadmodule attack is a User to Root attack against SunOS 4.1 systems that use the xnews window system. The loadmodule program within SunOS 4.1.x is used by the xnews window system server to load two dynamically loadable kernel drivers into the currently running system and to create special devices in the /dev directory to use those modules. Because of a bug in the way the loadmodule program sanitizes its environment, unauthorized users can gain root access on the local machine [8].

Simulation Details

The code for the loadmodule exploit script is widely available on the internet. This code is usually in the form of a shell script—it does not need to be compiled before it is run.

The steps of the attack are quite simple:

1. Change the value of the internal field separator (or IFS) variable to a slash.
2. Add "." To the front of the PATH variable
3. Copy "/bin/sh" to "./bin"
4. Execute "/usr/openwin/bin/loadmodule a".

When the loadmodule shell script (which is setuid root by default) executes, it attempts to run the command "exec('/bin/a');". Since the IFS variable has been changed to "/" the string "/bin/a" is parsed into two tokens, and the loadmodule script attempts to run the first—"bin". Since the attacker has conveniently put a copy of "/bin/sh" in the current directory and named it "bin", the loadmodule script (which is running as root) will exec "./bin"—giving the attacker a shell with root privileges.

Attack Signature

This attack can be identified either by performing bottleneck verification with a host-based intrusion detection system, or by keyword spotting with a network based intrusion

detection system. A simple rule could say that any session which contained the strings "set \$IFS='V'" and "loadmodule" in close proximity was probably a loadmodule attack. Of course, an attacker could quite easily hide from such a simple rule. Detailed discussion of such methods for hiding is presented in Chapter 11.

7.5 Perl

U-b-S

Description

The Perl attack is a User to Root attack that exploits a bug in some Perl implementations. Suidperl is a version of Perl that supports saved set-user-ID and set-group-ID scripts. In early versions of suidperl the interpreter does not properly relinquish its root privileges when changing its effective user and group IDs. On a system that has the suidperl, or sperl, program installed and supports saved set-user-ID and saved set-group-ID, anyone with access to an account on the system can gain root access [12].

Simulation Details

A perl script that uses this vulnerability to gain root access was made publicly available in August 1996 [51]. The code is only two lines long, and can easily be executed from the command-line. Once this perl script has run, the user will be presented with a new shell that is running with root privileges.

Attack Signature

The methods by which an intrusion detection system could identify a perl exploit attempt are identical to those described above for the loadmodule attack. A host-based intrusion detection system could notice that a root shell was spawned without a legal user to root transition, or a network-based intrusion detection system could look the strings "\$>=0; \$<=0;" or "exec ('/bin/sh');", which have little valid use except in an exploit attempt.

7.6 Ps

U-b-S

Description

The Ps attack takes advantage of a race condition in the version of "ps" distributed with Solaris 2.5 and allows an attacker to execute arbitrary code with root privilege. This race condition can only be exploited to gain root access if the user has access to the temporary files. Access to temporary files may be obtained if the permissions on the /tmp and /var/tmp directories are set incorrectly. Any users logged in to the system can gain unauthorized root privileges by exploiting this race condition [9].

Simulation Details

This exploit is possible because of a combination of the ps program not carefully managing temporary files and a buffer overflow. A shell script that builds a carefully constructed temporary file, creates a C-file, compiles the code and executes the exploit was found at Rootshell.com [52]. Once an attacker has transfers this shell script onto a Solaris victim machine and runs it, a root shell will be spawned for the attacker.

Attack Signature

Methods for finding this attack are essentially the same as the methods for finding the eject, ffbconfig, or fdformat attacks.

7.7 Xterm

U-b-S

Description

The Xterm attack exploits a buffer overflow in the Xaw library distributed with Redhat Linux 5.0 (as well as other operating systems not used in the simulation) and allows an attacker to execute arbitrary instructions with root privilege. Problems exist in both the xterm program and the Xaw library that allow user supplied data to cause buffer overflows in both the xterm program and any program that uses the Xaw library. These buffer overflows are associated with the processing of data related to the inputMethod and preeditType resources (for both xterm and Xaw) and the *Keymap resources (for xterm). Exploiting these buffer overflows with xterm when it is installed setuid-root or with any setuid-root program that uses the Xaw library can allow an unprivileged user to gain root access to the system [21].

Simulation Details

C source code that exploits this vulnerability on Redhat Linux 5.0 systems was found at the Rootshell website [56]. Once again, an attacker can compile this C code, and when the resulting program is run the attacker is given a shell running with root privileges.

Attack Signature

Methods for finding this attack are essentially the same as the methods for finding the eject, ffbconfig, or fdformat attacks.

Chapter 8

Remote to User Attacks

A Remote to User attack occurs when an attacker who has the ability to send packets to a machine over a network—but who does not have an account on that machine—exploits some vulnerability to gain local access as a user of that machine. There are many possible ways an attacker can gain unauthorized access to a local account on a machine. Some of the attacks discussed within this section exploit buffer overflows in network server software (imap, named, sendmail). The Dictionary, Ftp-Write, Guest and Xsnoop attacks all attempt to exploit weak or misconfigured system security policies. The Xlock attack involves social engineering—in order for the attack to be successful the attacker must successfully spoof a human operator into supplying their password to a screensaver that is actually a trojan horse. Figure 8-1 summarizes the characteristics of the Remote to User attacks that were included in the 1998 DARPA intrusion detection evaluation. The following sections provide details of each of these attacks.

8.1 Dictionary

R-a-U

Description

The Dictionary attack is a Remote to Local User attack in which an attacker tries to gain access to some machine by making repeated guesses at possible usernames and passwords.

Name	Service	Vulnerable Platforms	Mechanism	Time to Implement	Effect
Dictionary	telnet, rlogin, pop, imap, ftp	All	Abuse of Feature	Medium	User-level access
Ftp-write	ftp	All	Misconfiguration	Short	User-level access
Guest	telnet, rlogin	All	Misconfiguration	Short	User-level access
Imap	imap	Linux	Bug	Short	Root Shell
Named	dns	Linux	Bug	Short	Root Shell
Phf	http	All	Bug	Short	Execute commands as user http
Sendmail	smtp	Linux	Bug	Long	Execute commands as root
Xlock	X	All	Misconfiguration	Medium	Spoof user to obtain password
Xsnoop	X	All	Misconfiguration	Short	Monitor Keystrokes remotely

Figure 8-1: Summary of Remote to Local Attacks

Users typically do not choose good passwords, so an attacker who knows the username of a particular user (or the names of all users) will attempt to gain access to this user's account by making guesses at possible passwords. Dictionary guessing can be done with many services; telnet, ftp, pop, rlogin, and imap are the most prominent services that support authentication using usernames and passwords. Figure 8-2 is a plot of the connections made to the pop3 port of a victim machine during a dictionary attack that is using the pop service to check for valid login/password combinations. The horizontal axis of this plot represents time in minutes, and each line segment in the plot is a single connection to the pop3 service. Lines representing successive sessions are

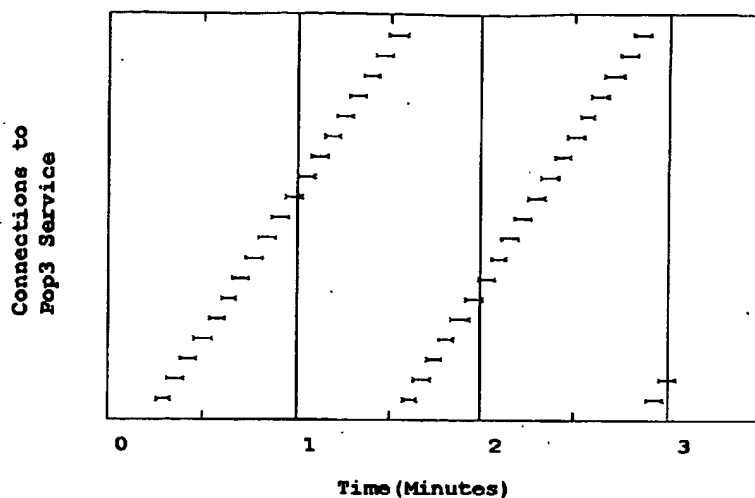


Figure 8-2: Plot of Connections to Pop3 Service During a Dictionary Attack

displaced vertically slightly and wrap around (in this figure at roughly 1.5 minutes). The length of the lines represents the length of the pop sessions. Lines begin with a greater-than sign ">" and end with a less-than sign "<", and thus form an "x" for short sessions. In all, this example dictionary attack consists of 40 attempts to log in, with a 4 second delay between each attempt.

Simulation Details

A perl script that performed automated password guessing on a variety of services was developed specifically for use in our evaluation. The "Netguess" perl script could take in a file of possible username/password combinations, or create guesses for the password based on simple permutations of the username. Within the simulation, this script was used to perform between 10 and 100 login attempts on the telnet, ftp, and pop services.

When the script was successful in gaining access to the system, it would immediately quit and report success.

Attack Signature

An intrusion detection system that finds attempted dictionary attacks needs to know the session protocol of every service that provides username/password authentication. For a given service, the intrusion detection system must be able to recognize and record failed login attempts. Once this functionality is available, detecting dictionary attacks is a matter of setting a detection threshold based on the number of failed login attempts within a given period of time.

8.2 Ftp-write

R-c-U

Description

The Ftp-write attack is a Remote to Local User attack that takes advantage of a common anonymous ftp misconfiguration. The anonymous ftp root directory and its subdirectories should not be owned by the ftp account or be in the same group as the ftp account. If any of these directories are owned by ftp or are in the same group as the ftp account and are not write protected, an intruder will be able to add files (such as an rhosts file) and eventually gain local access to the system [7].

Simulation Details

This attack was implemented as an expect script which was created explicitly for use in the simulation. This expect script anonymously logged in to the ftp service on the victim machine, created a ".rhosts" file with the string "+ +" in it within the ftp home directory, disconnected from the ftp server, used rlogin to connect back to the server as user "ftp", and finally performed some actions on the victim machine. Creating a ".rhosts" file in the ftp home directory with the entry "+ +" in it allows any user from any machine to rlogin to the victim as user "ftp".

Attack Signature

An intrusion detection system can monitor for this attack by watching all anonymous ftp sessions and assuring that no files are created in the ftp root directory.

8.3 Guest

R-c-U

Description

The Guest attack is a variant of the Dictionary attack described in Section 8.1. On badly configured systems, guest accounts are often left with no password or with an easy to guess password. Because most operating systems ship with the guest account activated by default, this is one of the first and simplest vulnerabilities an attacker will attempt to exploit [27].

Simulation Details

The Guest attack is a simplified version of the Dictionary attack discussed earlier in this chapter. The same "Netguess" perl script that was used to simulate a Dictionary attack was used to simulate the Guest attack—the only difference between the implementation of the two attacks was the command-line options that were passed to the Netguess program. Whereas the Dictionary attack would try up to a hundred user names and thousands of username/password combinations, the Guest attack would make only a couple of login attempts, using combinations such as "guest/<none>", "guest/guest", "anonymous/<none>" and "anonymous/anonymous".

Attack Signature

Because the Guest attack is essentially a subset of the Dictionary attack, the methods for finding the two attacks are basically the same. An intrusion detection system that is looking for a Dictionary attack already should need only minor tuning in order to find attempts to log in to the guest account.

8.4 Imap

R-b-S

Description

The Imap attack exploits a buffer overflow in the Imap server of Redhat Linux 4.2 that allows remote attackers to execute arbitrary instructions with root privileges. The Imap server must be run with root privileges so it can access mail folders and undertake some file manipulation on behalf of the user logging in. After login, these privileges are discarded. However, a buffer overflow bug exists in the authentication code of the login transaction, and this bug can be exploited to gain root access on the server. By sending carefully crafted text to a system running a vulnerable version of the Imap server, remote users can cause a buffer overflow and execute arbitrary instructions with root privileges [16].

Simulation Details

The Imap attack used in the 1998 DARPA intrusion detection evaluation was part of the Impack 1.03 attack toolkit [34]. This toolkit contained precompiled binary programs for the Linux platform that would scan for vulnerable machines, as well as send the necessary message to exploit the buffer overflow and gain access to a root shell. The Impack contained detailed instructions on how to use these precompiled programs and took very little skill to use. The release of the Impack made this vulnerability especially dangerous, as any user with a Linux machine and the ability to follow instructions could use this attack to remotely gain root access to any vulnerable hosts.

Attack Signature

The Imap attack can be identified by an intrusion detection system that has been programmed to monitor network traffic for oversized Imap authentication strings.

8.5 Named

R-b-S

Description

The Named attack exploits a buffer overflow in BIND version 4.9 releases prior to BIND 4.9.7 and BIND 8 releases prior to 8.1.2. An improperly or maliciously formatted inverse query on a TCP stream destined for the named service can crash the named server or allow an attacker to gain root privileges [19].

Simulation Details

The version of the Named exploit used in the simulation was adapted from a C program originally posted to the Bugtraq mailing list. This program, once compiled, would connect to the named port on a victim machine and overflow a buffer of the named server with instructions that would send an xterm running with root privilege back to the attacker's X console. Because this attack involved interaction with X, all of the Named attacks included in the simulation were run by human actors.

Attack Signature

The Named attack can be identified by watching DNS inverse query requests for messages that are longer than the 4096 byte buffer allocated for these requests within the "named" server.

8.6 Phf

R-b-U

Description

The Phf attack abuses a badly written CGI script to execute commands with the privilege level of the http server. Any CGI program which relies on the CGI function `escape_shell_cmd()` to prevent exploitation of shell-based library calls may be vulnerable to attack. In particular, this vulnerability is manifested by the "phf" program that is distributed with the example code for the Apache web server [11].

Simulation Details

The Phf attack is quite simple to implement because it requires only the ability to connect to a network socket and issue an http request. Within the simulation, the Netcat [31] program was used to generate this http request. Although this vulnerability allows an attacker to run any command on the server, the command used throughout the simulation was `"/bin/cat /etc/passwd"`. Using the Phf attack with this command reveals the contents of the victim system's password file to users with no account on the victim machine.

Attack Signature

To find the Phf attack, an intrusion detection system can monitor http requests watching for invocations of the phf command with arguments that specify commands to be run. Examples of commands that an attacker might attempt to execute by exploiting the phf exploit are: `cat /etc/passwd`, `id`, `whoami`, or `xterm`.

8.7 Sendmail

R-b-S

Description

The Sendmail attack exploits a buffer overflow in version 8.8.3 of sendmail and allows a remote attacker to execute commands with superuser privileges. By sending a carefully crafted email message to a system running a vulnerable version of sendmail, intruders can force sendmail to execute arbitrary commands with root privilege [15].

Simulation Details

Although this vulnerability was widely known about at the time of the evaluation, no code that exploited this vulnerability had been posted to public forums such as Bugtraq, or Rootshell.com. A significant period of time (one person working for more than two weeks) was spent developing the first known implementation of this exploit explicitly for use in the evaluation. The implementation consists of a carefully constructed mail message which, when sent to the victim machine with a vulnerable version of sendmail, adds a new entry with root privilege to the end of the password file on the victim system. Once this new entry has been added, the attacker can log into the machine as this new user and execute commands as a root user. Figure 8-3 provides an illustration of an instantiation of the Sendmail attack as it was implemented in the simulation. In step 1 of this illustration, the attacker sends a carefully crafted e-mail message to the victim machine. In step 2, the sendmail daemon starts to process this message, overflows one of its buffers, and executes the attacker's inserted commands that create a new entry in the password file. In step 3, the attacker comes back to the victim machine and uses the new password file entry to gain root access to the victim machine and perform some malicious actions.

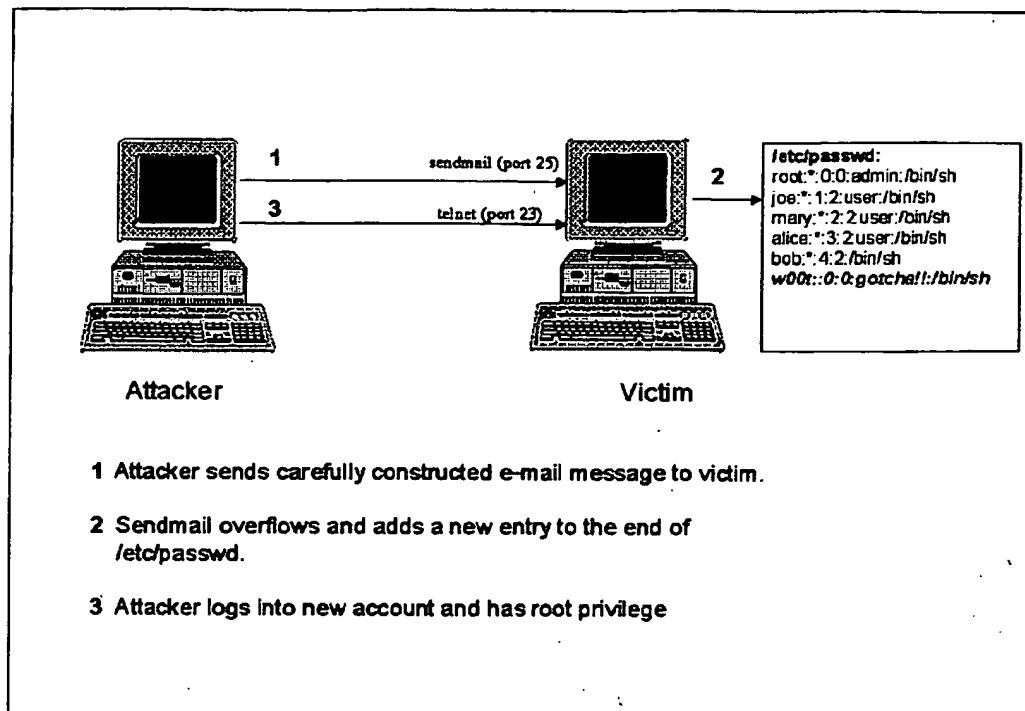


Figure 8-3: Illustration of the Sendmail Attack.

Attack Signature

The Sendmail attack overflows a buffer in the MIME decoding routine of the sendmail program. In order for an intrusion detection system to identify a Sendmail attack it must monitor all incoming mail traffic and check for messages that contain a MIME header line that is inappropriately large.

8.8 Xlock

R-cs-Intecept(Keystrokes)

Description

In the Xlock attack, a remote attacker gains local access by fooling a legitimate user who has left their X console unprotected, into revealing their password. An attacker can display a modified version of the xlock program on the display of a user who has left their X display open (as would happen after typing "xhost +"), hoping to convince the user sitting at that console to type in their password. If the user sitting at the machine being attacked actually types their password into the trojan version of xlock the password will be sent back to the attacker.

Simulation Details

This attack was created specifically for use in the evaluation. A special version of the xlock program was created by making small modifications to the publicly available source code for the xlock program. The standard version of xlock will not display on a remote display, and does not save or output the password that the user enters. A modified version of xlock was created that allows the attacker to display the screen saver on a remote display and returns the password entered by the victim. Because of the required interaction with the X server, the Xlock attack was always run by a human actor.

Attack Signature

Two factors make this attack quite difficult for an intrusion detection system to identify. First, this attack is a spoofing attack that does not abuse a bug in the system, but relies on assumptions by the person who is currently using the system. Second, this attack is embedded in the X protocol, and an intrusion detection system that is trying to identify this attack must understand and parse these X communications in order to identify the

Xlock attack. One non-optimal way of dealing with these difficulties, is to program an intrusion detection system to identify as suspicious any X traffic that is originating from an unknown machine that is destined for a machine being monitored.

8.9 Xsnoop

R-c-Intercept(Keystrokes)

Description

In the Xsnoop attack, an attacker watches the keystrokes processed by an unprotected X server to try to gain information that can be used gain local access the victim system. An attacker can monitor keystrokes on the X server of a user who has left their X display open. A log of keystrokes is useful to an attacker because it might contain confidential information, or information that can be used to gain access to the system such as the username and password of the user being monitored.

Simulation Details

The Xsnoop program used in the simulation was adapted from C code originally posted to the Bugtraq mailing list. The Xsnoop program runs locally on the attacker's machine. The program connects to the victim's X Server and requests to be notified of all X KeyPress Events. If the attacker has permission to make this request (as would happen if the victim had typed "xhost +", or if the victim's X Server is configured to allow connections from all hosts by default) the Xsnoop program will be sent all KeyPress events that occur on the victim X Server. The Xsnoop program then uses the KeyPress events to provide the attacker with a view of all the keystrokes the victim. Because this attack required interaction with the X Server, it was always run by a human actor.

Attack Signature

An network based intrusion detection system can identify the Xsnoop attack by parsing the X protocol information in packets destined for remote X clients and noting that X KeyPress events are being transmitted to a remote machine. Because the Xsnoop attack results from bad security policy (leaving an X Server unsecured) and not simply a bug,

the presence of these packets alone does not signify that an attack is taking place. The intrusion detection system must know whether the security policy of the machine being monitored allows unauthenticated X connections from anywhere.

Chapter 9

Probes

In recent years, a growing number of programs have been distributed that can automatically scan a network of computers to gather information or find known vulnerabilities [27]. These network probes are quite useful to an attacker who is staging a future attack. An attacker with a map of which machines and services are available on a network can use this information to look for weak points. Some of these scanning tools (satan, saint, mscan) enable even a very unskilled attacker to very quickly check hundreds or thousands of machines on a network for known vulnerabilities. Figure 9-1 provides a summary of the probes that are discussed in the remainder of this chapter. The

Name	Service	Vulnerable Platforms	Mechanism	Time to Implement	Effect
Ipsweep	ICMP	All	Abuse of Feature	Short	Finds active machines
Mscan	many	All	Abuse of Feature	Short	Looks for known vulnerabilities
Nmap	many	All	Abuse of Feature	Short	Finds active ports on a machine
Saint	many	All	Abuse of Feature	Short	Looks for known vulnerabilities
Satan	many	All	Abuse of Feature	Short	Looks for known vulnerabilities

Figure 9-1: Summary of Probes

following sections describe in detail each of the probes that was used in the 1998 DARPA intrusion detection evaluation.

9.1 Ipsweep

R-a-Probe(Machines)

Background

An Ipsweep attack is a surveillance sweep to determine which hosts are listening on a network. This information is useful to an attacker in staging attacks and searching for vulnerable machines.

Simulation Details

There are many methods an attacker can use to perform an Ipsweep attack. The most common method—and the method used within the simulation—is to send ICMP Ping packets to every possible address within a subnet and wait to see which machines respond. The Ipsweep probes in the simulation were not stealthy—the sweeps were performed linearly, quickly and from a single source.

Attack Signature

An intrusion detection system looking for the simple Ipsweep used in the simulation can look for many Ping packets, destined for every possible machine on a network, all coming from the same source.

9.2 Mscan

R-a-Probe(Known Vulnerabilities)

Description

Mscan is a probing tool that uses both DNS zone transfers and/or brute force scanning of IP addresses to locate machines, and test them for vulnerabilities [20].

Simulation Details

The Mscan program used in the simulation was compiled from source code found at [57]. Mscan was easy to run and has several command line options for specifying the number of machines to scan and which vulnerabilities to look for. Within the simulation, mscan was used to scan the entire eyrie.af.mil domain for the following vulnerabilities: statd, imap, pop, IRIX machines that have accounts with no passwords, bind, various cgi-bin vulnerabilities, NFS, and open X servers.

Attack Signature

The signature of this attack will vary depending on which vulnerabilities are being scanned for and how many machines are being scanned. In general, an intrusion detection system can find an Mscan attack by looking for connections from a single outside machine to the ports listed above on one or more machines within a short period of time.

9.3 Nmap

R-a-Probe(Services)

Description

Nmap is a general-purpose tool for performing network scans. Nmap supports many different types of portscans—options include SYN, FIN and ACK scanning with both TCP and UDP, as well as ICMP (Ping) scanning [45]. The Nmap program also allows a user to specify which ports to scan, how much time to wait between each port, and whether the ports should be scanned sequentially or in a random order.

Simulation Details

At the time of the evaluation, Nmap was the most complete publicly available scanning tool. During the simulation, Nmap was used to perform portscans on between one and ten computers using SYN scanning, FIN scanning, and UDP scanning of victim machines. Both sequential and random scans were performed in the simulation, and the timeout between packets was varied to be anywhere from one second to six minutes. The number of ports scanned on each machine was varied between three and one thousand.

Attack Signature

The signature of a portscan using the Nmap tool varies widely depending on the mode of operation selected. Despite this variance of modes, all portscans share some common features. A portscan can be recognized by noting that network packets (whether via TCP or UDP, or via only FIN packets or only SYN packets) have been sent to several (or more) ports on a victim or group of victims within some window of time. Two factors complicate the identification of portscans. First, a portscan can happen very slowly. An attacker who is patient could probe one port per day. Current intrusion detection systems do not keep enough state to recognize a portscan happening over such a long period of

time. With the amount of network traffic sent over a typical network, keeping enough active state within the intrusion detection system to recognize one connection per day for one hundred days as a one hundred day long portscan is simply not practical. Second, the connections don't necessarily all have to come from the same host. A group can perform a coordinated scan with each member scanning only a subset of machines or ports. By combining these methods, a group could perform a "low/slow" portscan that would be very hard to recognize.

9.4 Saint

R-a-Probe(Known Vulnerabilities)

Description

SAINT is the Security Administrator's Integrated Network Tool. In its simplest mode, it gathers as much information about remote hosts and networks as possible by examining such network services as finger, NFS, NIS, ftp and tftp, rexd, statd, and other services. The information gathered includes the presence of various network information services as well as potential security flaws. These flaws include incorrectly setup or configured network services, well-known bugs in system or network utilities, and poor policy decisions. Although SAINT is not intended for use as an attack tool, it does provide security information that is quite useful to an attacker [58].

Simulation Details

SAINT is distributed as a collection of perl and C programs and is known to run on Solaris, Linux, and Irix systems. Within the simulation, the Saint program was run from a Linux traffic generator and was used to probe several victim machines for vulnerabilities. SAINT's behavior is controlled by a configuration file which allows the user to specify several parameters. The most important parameters are the list of machines to scan, and how heavily to scan these machine (light, normal, or heavy). In light mode, SAINT will probe the victim for dns and rpc vulnerabilities and will look for unsecured NFS mount points. In normal mode SAINT will also check for vulnerabilities in fingerd, rusersd, and bootd, and will perform a portscan on several tcp (70, 80,ftp, telnet, smtp, nntp, uucp) and udp (dns, 177) ports. Heavy mode is the same as normal mode except that many more ports are scanned.

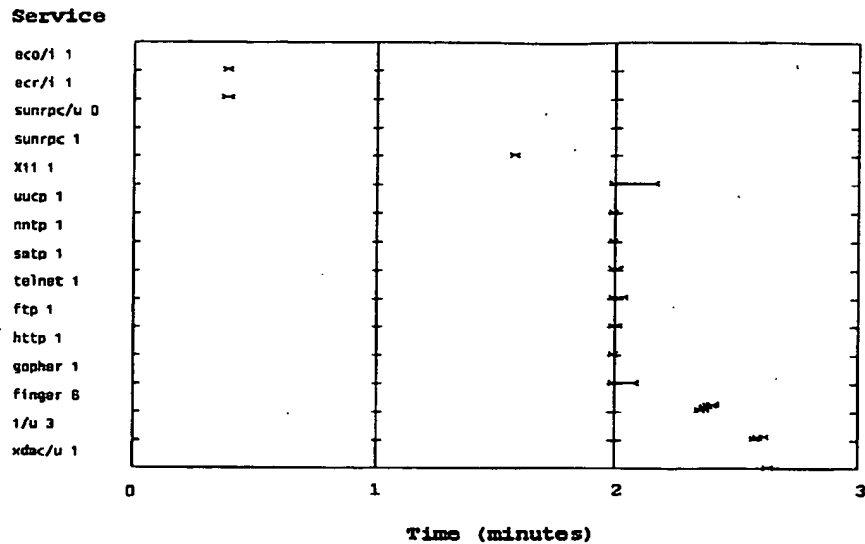


Figure 9-2: Plot of Connections During a Medium Level Saint Scan

Attack Signature

A Saint scan of a network leaves a distinct signature that will vary depending on the level of scanning being performed. The Saint program performs each scan in a nearly deterministic fashion. To identify a Saint scan, an intrusion detection system needs to be able to recognize the distinct set of network traffic the scan creates. Figure 9-2 is a plot that provides a graphical view of this signature. The horizontal axis of this plot represents time in minutes, and the different services probed are presented along the vertical axis. The names of the services are shown on the left side of this plot, and connections for each service are plotted within each named region. The numbers after the service names are the number of separate tcp connections or of udp or icmp packets. Names ending in “/i” indicate that packets use the icmp protocol and names ending in “/u” indicate that packets use the udp protocol. All other services use the tcp protocol.

Each line segment represents a connection to a service. This plot shows the unique signature of a medium level Saint scan. Because this signature does not change significantly across multiple instantiations of the Saint attack, an intrusion detection system that has been trained to recognize the pattern of connections shown in Figure 9-2 will probably detect other Saint attacks.

9.5 Satan

R-a-Probe(Known Vulnerabilities)

Description

SATAN is an early predecessor of the SAINT scanning program described in the last section. While SAINT and SATAN are quite similar in purpose and design, the particular vulnerabilities that each tool checks for are slightly different [24].

Simulation Details

Like SAINT, SATAN is distributed as a collection of perl and C programs that can be run either from within a web browser or from the UNIX command prompt. SATAN supports three levels of scanning: light, normal, and heavy. The vulnerabilities that SATAN checks for in heavy mode are:

- NFS export to unprivileged programs
- NFS export via portmapper
- NIS password file access
- REXD access
- tftp file access
- remote shell access
- unrestricted NFS export
- unrestricted X Server access
- write-able ftp home directory
- several Sendmail vulnerabilities
- several ftp vulnerabilities

Scans in light and normal mode simply check for smaller subsets of these vulnerabilities.

Attack Signature

A SATAN scan of a network can be recognized by the consistent pattern of network traffic the program creates. The checks for the vulnerabilities listed above are always performed in the same order.

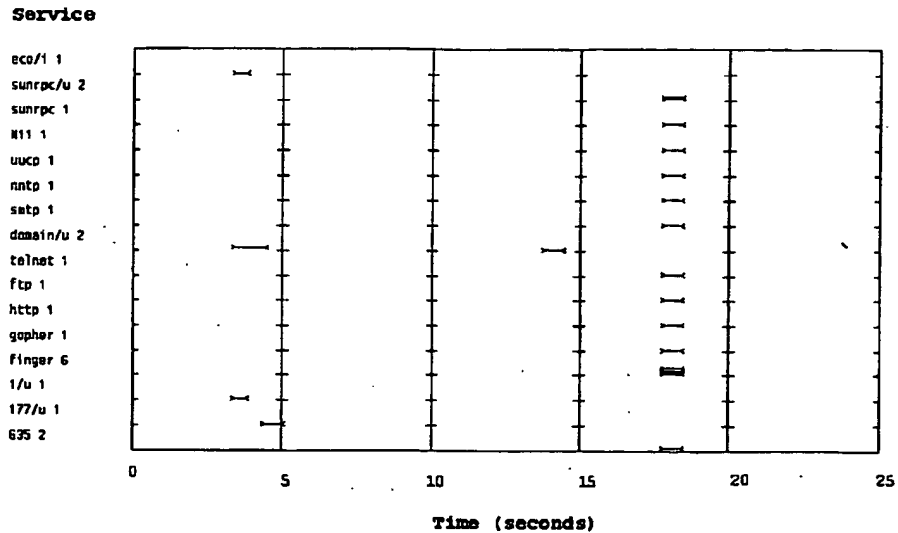


Figure 9-3: Plot of Connections During a Medium Level Satan Scan

Figure 9-3 shows a plot of the services probed during an example medium level Satan scan. The horizontal axis of this plot represents time in seconds and the various services that are probed are presented on the vertical axis. Each line segment in the plot represents a single connection to a service. Every medium level Satan scan will have a signature very similar to that shown in Figure 9-3.

Chapter 10

Realistic Intrusion Scenarios

An attack against a computer system can consist of several phases. An attacker might run several probes against a system looking for weaknesses, then use information gained from these probes to exploit some security vulnerability and gain access to the system. Later an attacker can use this access to install a backdoor in the system, or plant a “logic bomb” in some program that causes damage at some point in the future. The actions of a single intruder might be spread out over the course of weeks or even months. Some attackers make no attempt to hide their intrusions from systems administrators while others are very careful to cover their tracks. In order to accurately test an intrusion detection system’s effectiveness in finding real attackers, it was necessary to not only develop a database of realistic exploits and probes, but to combine these exploits and probes with probable attacker actions and realistic goals to generate complete attack scenarios. This chapter describes how these various parameters of attacker behavior were varied to create a realistic simulation of the wide variety of attacker behavior that is manifested by actual intrusions.

10.1 Attack Scenarios

Most of the attacks that were included in the evaluation consist of a single session, or a few sessions that all occur within some short period of time. In the real world, an

attacker often has a goal in mind, and sometimes this goal cannot be achieved in a single session. Several complex scenarios were added to the collected data in an attempt to create a better simulation of real attacker behavior. Each scenario consists of a planned sequence of sessions (sometimes over the course of a week or more) that represent the actions of a single individual in pursuit of a goal. The following subsections describe each scenario in detail.

10.1.1 Cracker

The default scenario that occurred in 95% of sessions is that a curious cracker was trying to gain access to a machine just to prove that it could be done. Usually these crackers are simply trying to break into as many machines as they can, and may install a backdoor or download the password file in order to guarantee that they can access the machine again. Crackers are represented as individuals of different skill levels, some perform all of their actions in the clear, while others are aware that an intrusion detection system is present and take actions to avoid detection.

10.1.2 Spy

The spy is an information collector who comes back to a compromised machine several times to collect information. A spy might be looking for confidential data files or reading user's personal mail. A spy will take steps to minimize the possibility of detection.

10.1.3 Rootkit

The rootkit scenario could be viewed as an extension of the default Cracker scenario.

A rootkit is a collection of programs that are intended to help a hacker maintain access to a machine once it has been compromised. A typical rootkit consists of a sniffer, versions of login, su, and other programs with backdoors which allow for access, and new versions of ps, netstat, and ls that hide the fact that a sniffer is running and hide files in certain directories. Once the rootkit has been installed, the attacker comes back several times to download the sniffer logs.

10.1.4 Http Tunnel

The Http Tunnel scenario was originally developed as a method of defeating a firewall and for continuing to access a system while minimizing the chance of detection. The http tunnelling tools used in this scenario were developed specifically for use in this evaluation. Figure 10-1 provides an overview of this scenario. Assuming that a hacker is able to penetrate a firewall once (perhaps by sending an email with the executable content of the client in it or by dialing into a modem) a server program can be installed that masquerades as a normal user browsing web pages. Once this server has been installed, the hacker can issue requests to execute commands or transfer files with interaction happening in web traffic between the server and the hacker. For the simulation, the data was exchanged through cookies that rode along with a web request. This general method is very flexible however, and the data could have been tunneled using any cryptographic or steganographic technique. Tunneling through http was chosen with hopes that the large amount of http traffic most networks see in a typical day would obscure the actions of the attacker.

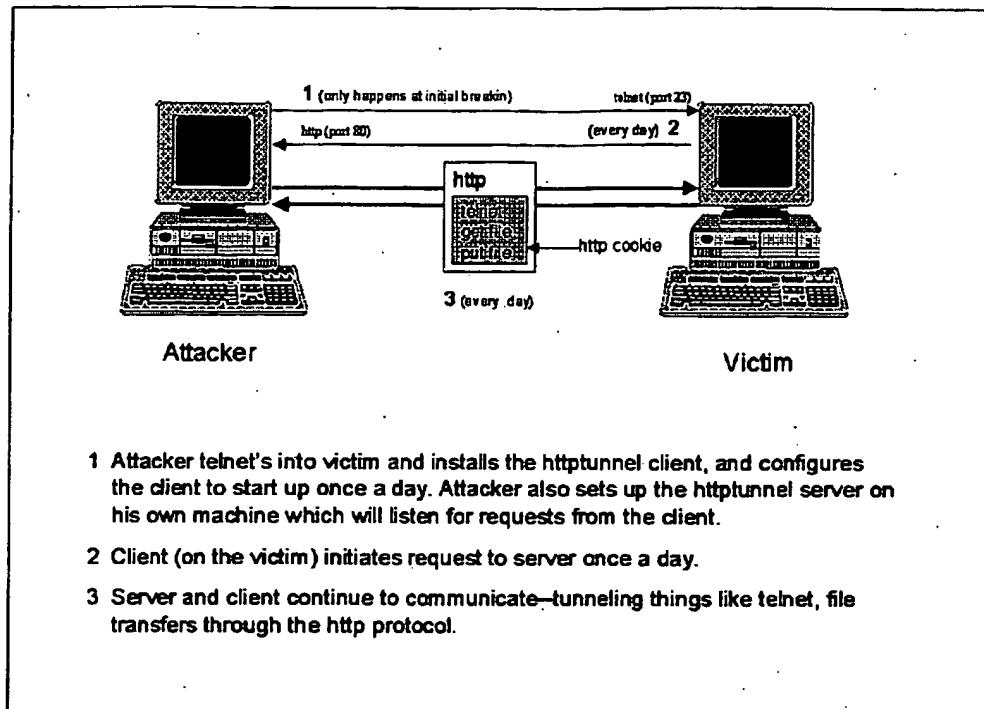


Figure 10-1: Attacker Uses Http to Tunnel Through a Firewall

10.1.5 SNMP Monitoring

An attacker who has guessed the SNMP community password of a router will then be able to monitor the traffic levels on that router, and may be able to issue commands to the router to change default routes or allow connections from a previously forbidden host or network.

10.1.6 Multihop

Some intrusion detection systems monitor traffic immediately outside of a router and only see traffic going into or coming out of that network. The multihop scenario was designed to test whether these systems could find attacks where an attacker first breaks into one inside machine, and then uses this inside machine for further attacks on the rest

of the machines on the network. This would be a very effective means of launching a Denial of Service attack undetected, as an intrusion detection system which sits just outside the network being monitored would not see a Denial of Service attack that originated from the internal network.

10.1.7 Disgruntled/Malicious User

These sessions simulate an attacker who is not interested in collecting information from a system or gaining access to a system, but is simply interested in doing damage. This is a common threat in operational computer networks [32]. Within the simulation, one malicious user re-formatted the primary disk partition of a victim machine.

Chapter 11

Stealthiness and Actions

In addition to varying the methods and intentions of the simulated attackers, attention was given to the extent to which attackers tried to hide their actions from either an individual who is monitoring the system, or an intrusion detection system. There are several ways that attackers can reduce their chances of being detected by the administrator of a network. Skilled attackers might try to cover their tracks by editing system logs or resetting the modification date on files that they replaced or modified. These actions are generally intended to reduce the chance of detection by a human administrator. Attackers may also be aware that an intrusion detection system is monitoring a network, and may try to hide from the intrusion detection system as well. Methods for being stealthy vary depending on the type of attack.

11.1 Avoiding Detection of Denial of Service (R-Deny)

Denial of Service attacks are difficult to make stealthy. One method an attacker can use to hide a denial of service attack is to gain the cooperation of a large group and break up the attack so pieces of it are coming from several different sources. Another method an attacker can use is to send thousands of packets with different spoofed source-addresses. Sending these spoofed packets will not make identifying the attack any harder, but they will make it more difficult to track down and stop the attacker because the victim has no

way of knowing which of the thousands of addresses the actual attack is coming from. Both of these methods do not make the attack any harder to detect, but simply reduce the chances that the attacker will be caught. No stealthy denial of service attacks were included in the 1998 DARPA intrusion detection evaluation.

11.2 Avoiding Detection of Probes (R-Probe)

Several methods can be used either to hide the fact that a probe is occurring, or obscure the identity of the party who is performing the probe. The following paragraphs describe methods of increasing stealth that were used in the probes included in the 1998 DARPA intrusion detection evaluation.

Scan Slowly and Randomly: If an attacker wants to hide the fact a probe is occurring, the probe can be configured to occur slowly and probe ports or machines in a non-linear order. An intrusion detection system will have a very hard time identifying one stray connection per hour to a random port as a port sweep initiated by an attacker.

Probe With Half-Open or Other Unlogged Connections: Another method of scanning stealthily is to probe with half-open connections. A connection for which the three-way TCP handshake is never completed will not be logged by the operating system. There are several tools available that will perform this type of half-open (FIN) scanning of a network.

Use an Intermediate Machine to Obscure the Real Source of the Scan: One way attackers can hide their identity is to use an ftp bounce probe. Some ftp servers will allow anyone to tell them to send data to a particular port on a particular machine. An attacker can look at the response the ftp server gives from such a request and ascertain whether that port is listening on the victim machine. The portscan will appear to be

coming from an anonymous ftp server, and this simple step may be enough to assure that the party who is really doing the scanning is never identified.

11.3 Avoiding Detection of User to Root (L-?-S) Attacks

There are many ways that User to Root attacks can be made stealthy. The following paragraphs discuss each of the methods that were used to make a number of the User to Root attacks in the 1998 DARPA intrusion detection evaluation stealthy.

Keyword Hiding: Some intrusion detection systems that attempt to detect illegal User to Root transitions rely on keyword spotting to detect intruders. For example, if the system observes the text of the C source code for the publicly available Eject exploit in a telnet or rlogin session, it will flag this session as being suspicious. By uuencoding or Mime encoding the text of the code before sending it over the network connection, an attacker would avoid detection by such a system.

Output Hiding: It is possible to identify attacks by looking at the output that is displayed on the terminal when the exploit is run. An attacker can avoid detection via this mechanism by sending all the output of commands that are run to a file and encoding (again with some method like ROT13, uuencode, or gzip) the file before displaying or transferring it.

Command Hiding: A system might also look for an attacker to run some command that only the superuser should be able to run, such as displaying the contents of the shadow password file. The invocation of a command that the attacker wishes to hide from someone who is looking for certain suspicious commands or actions can be obfuscated by using glob constructs and character replacement. Instead of typing the command "cat /etc/passwd", the attacker can issue the command "[r,s,t,b]?[l,w,n,m]/[c,d]?t

`/?[c,d,e]/*a?s*`". When the shell tries to interpret this input string it will do replacement of the glob characters and find that the only valid match for this string is `"/bin/cat /etc/passwd"`.

Delayed Attack: An attacker can also avoid detection (or at least reduce the chances of identification), by separating the time of exploit from the initial time of access. This can be accomplished by submitting a job to the "at" or "cron" daemon which will run the attack at some later time. An attacker can log in as a normal user (which would not be noticed by an intrusion detection system) and submit a shell script to the "at" daemon which would execute any actions that the attacker desired three weeks (or six months, or five years!) after the attacker initially logged into the system. Figure 11-1 provides a graphical illustration of a delayed attack scenario. This scenario is similar to one that was used in the evaluation. First, at time segment 1 in the figure, the attacker uses a sniffed password to log into the victim machine as a normal user. This action will not be noticed by an intrusion detection system. Before logging out, the attacker submits a job to the cron or at daemon that performs a User to Root exploit and some malicious actions at some point in the future. At time segment 2 in the figure, the submitted script is run and the exploit occurs. Later, at time segment 3 the attacker can come back and log into the machine as a normal user to collect the output from the commands run during the second segment, again without being noticed. The only step in this process that is easily detectable by an intrusion detection system is step 2 when the actual exploit occurs. However, during this time segment the attacker is not logged in to the system. In order to successfully identify this attack, an intrusion

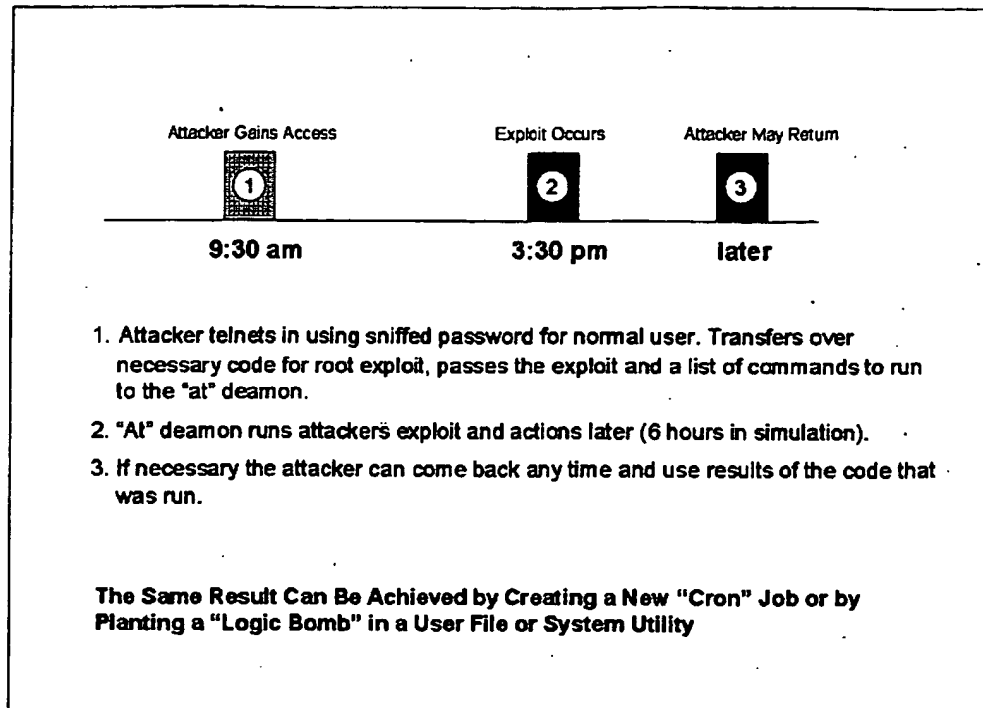


Figure 11-1: Attacker Can Use "at" to Temporally Dissociate the Time of Access and Time of Attack

detection system would need to correlate the earlier session to the malicious actions that occur when the attacker is not logged in.

Figure 11-2 is a transcript from an actual stealthy eject attack used in the simulation. This attack session contains the keyword hiding, output hiding, and command hiding techniques discussed above. Lines 10 through 16 contain the output of the attacker transferring the exploit files to the victim system. The files have been uuencoded to help avoid detection of the attack by a system that is looking for keywords that might be located in the text of the exploit files.

```

1 UNIX(r) System V Release 4.0 (pascal)
2
3 login: alie
4 Password:
5 Last login: Thu Jul 30 09:15:04 from 135.8.60.182
6 Sun Microsystems Inc. SunOS 5.5 Generic November 1995
7 pascal>
8 pascal> which gcc
9 /bin/gcc

```

```

10 pascal> uidecode
11 begin 644 /tmp/16256.c
12 M("@"@("J"B'@("I3;VQA<FES(#(N-2XQ("T@=&AI<R!E>!L;VET960@=V%S
13 M(&-O;7!!;&5D(&]N(%-O;&%R:7,R+C0@86YD(1E<W1E9"IO;@H@("@"BXU
14 '
15 end
16 <FILE TRANSFER TRUCATED FOR EXAMPLE>

```

**Keyword
Hiding**

```

17 pascal>
18
19 pascal> /bin/gcc -o /tmp/162562 /tmp/16256.c
20 pascal> which gcc
21 /bin/gcc
22 pascal> /bin/gcc -o /tmp/162563 /tmp/16256.c
23 pascal> uidecode
24 begin 644 /tmp/16256.c
25 M(VEN8VQU9&4@/5N:7-T9"YH/@IV:VED"FUA:6XH:6YT(&%R9V,L(&-H87(@
26 M"F%R9W9;72D*>PH@('E=)E=6ED*#L,"D"B'@97AE8VPH(B)B:6XO=&-S
27 0:"(L(GIC<V@B+#"LPI)"@*
28 '
29 end
30 pascal>
31
32 pascal> /tmp/162562
33 Jumping to address 0xeffff6a0 B[364] E[400] SO[704]
34 #

```

```

35 #ls | /u?r?l?c?l/b?n/g?ip | /b?n/u?e?c?d? gfd
36 begin 644 gfd
37 B*XL( (%@:P34 RM)+2XQY"H!D7K)8-J("P"DGT[% !
38
39 end
40 # /u?r?l?c?l/b?n/g?ip | /b?n/u?e?c?d? asd
41 begin 644 asd
42 M*XL( (%L:P34 RO-3+$U,COTT$C,R4S55$@<@T-##1*BU+BC454DN!?.-
43 /HOS$DTN CL7@K
44
45 end
46 # /bi?/c?/et/?p?s?w? | /u?r?l?c?l/b?n/g?ip | /b?n/u?e?c?d? hghhg
47 begin 644 hghhg
48 M*XL( (%P34 $Z:7 <QW7G{&(!@A 8@1" /ZP10%*R1.*;) C+UH( "%(
49 <FILE TRANSFER TRUCATED FOR EXAMPLE>
50 1 ?C^$O%+V/$3,:!)$

```

**Command
Hiding and
Output Hiding**

```

51
52 end
53 # exit
54 pascal> logout

```

Figure 11-2: Transcript of a Stealthy Eject Attack

In lines 34 through 54 the attacker has already gained root access and is now performing some malicious actions. The attacker uses both command hiding with glob constructs and output hiding using both uencode and gzip. For example, on line 46 the

attacker types “# /bi?/c? /et?/p?s?w? | /u?n?c?/b?n/g?ip | /b?n/u?e?c?d? hghhgf”, which looks like garbage, but is interpreted by the shell as “# /bin/cat /etc/passwd | /usr/local/bin/gzip | /bin/uueencode hghhgf”. Each of these techniques makes the attack harder to find for an intrusion detection system that is depending on recognition of key words for attack detection.

The User to Root attacks that were inserted into the data for the 1998 DARPA intrusion detection evaluation contained a mix of both stealthy and clear User to Root Attacks. For User to Root attacks in the four weeks of test data, 50 percent were performed in the clear with no actions, 25 percent were performed in the clear with actions, and 25 were performed stealthily with stealthy actions.

The stealthy methods presented in this section represent only a small subset of the possible steps an attacker could take in an attempt to make a User to Root attack less noticeable. Future evaluations will need to explore these additional methods more completely. As each generation of intrusion detection system learns how to defeat new stealthy methods, attackers will likely develop newer and better stealthy techniques.

11.4 Avoiding Detection of Remote to Local (R-?-L) Attacks

Many Remote to Local attacks have a unique signature that is hard to alter. In a dictionary-based password guessing attack the user simply must connect to a service many times and try different passwords. If an attacker wants to gain access to a machine by remotely overflowing the imap service they have to transmit the code that overflows the buffer across the network in the clear. For these reasons, there are fewer options for making Remote to Local attacks stealthy than there are for User to Root attacks. However, once an attacker has gained access and is interacting with a shell, all the same methods that were outlined in the User to Root section above can be used to hide further

actions the attacker might perform. None of the Remote to Local User attacks in the 1998 DARPA intrusion detection evaluation were designed to be stealthy.

11.5 Actions

Many of the attacks included within the data collected for the 1998 DARPA intrusion detection evaluation include some set of actions that were not directly related to the task of either gaining access or denying service. In some cases these actions were performed as part of a scenario where an attacker had a specific goal in mind. But, in many cases even attacks that were not developed with a specific scenario in mind contain a few actions that an attacker performs after gaining access. These actions were chosen from a set that was identified by looking at transcripts of actual intrusions and by considering the goals that an attacker might have. The following are descriptions of the actions that were performed during the simulation:

- **id** An attacker uses the "id" program to check whether an exploit obtains root access
- **cat /etc/password, cat /etc/shadow** Once the attacker has collected the contents of the password file the contents can be decrypted offline.
- **netstat** Netstat provides an attacker with a list of the network services a host offers.
- **mount, showmount** An attacker runs the mount or showmount command to look for NFS mounted drives that might provide access to another machine or additional data.
- **cat /etc/hosts.equiv** The "hosts.equiv" file contains a list of trusted machines. By adding new machines to this list of trusted hosts an attacker can assure later access.

- **creation of a suid root shell** By making a copy of /bin/sh or some other shell and setting its permissions to make the executable suid root the attacker can guarantee that they will later be able to obtain root access.
- **w, who** An attacker who is trying to avoid detection can use these commands to see who else is logged in to the system.
- **editing .rhosts** The .rhosts file is similar to /etc/hosts.equiv in that it specifies a list trusted hosts. If an attacker can add the string "+ +" to /root/.rhosts the system will allow anyone to perform remote commands (rsh, rlogin, rcp, etc.) as root on that system.
- **uname** The uname command gives an attacker information about the hardware and operating system of the victim host.
- **starting up new services** Attacker can start a new listening port for later access to the machine. They can start up the tftp service that will later allow them to read any world-readable file on the system without authenticating themselves. They can bind a second listening telnet daemon to some port other than the normal port (port 23) to get around a firewall. Or, attackers can set up the netcat program to listen on a port and give them a shell, or provide them with some piece of information whenever someone connects to that port and supplies a "magic word". More examples are available in [31].

Chapter 12

Attack Planning and Data Collection

Nine weeks of data were collected during the 1998 DARPA intrusion detection evaluation. This data consisted of normal traffic as well as inserted attacks. Of these nine weeks of data, the first seven weeks are considered to be the “training” dataset and the last two weeks are the “testing” dataset. After the first seven weeks of data was collected, the data and an answer key listing the time, name, and affected machines for each attack was distributed to participants. The participants then trained their intrusion detection systems on this data. After the training phase was complete, two additional weeks of data was distributed. Participants attempted to find attacks in this two weeks of data and returned a list of found attacks to Lincoln Laboratory. These lists of found attacks were then scored to determine the participant’s success rate and false alarm rate across various categories of attacks. The remainder of this chapter discusses the method used to select attacks and keep track of them, shows the actual numbers of attacks present in the 1998 evaluation, and describes the process for validating that an attack occurred.

12.1 Planning and Keeping Track of Attacks

A complete schedule of all attacks for the nine week period was developed by hand prior to the beginning of the data collection process. The number of instances of each of the 32 attack types was kept roughly equal. Once the schedule was created, the schedule was

stored in electronic form and was used to create the "exs" scripts that were used to run each attack. A collection of CGI scripts was written to allow the schedule to be conveniently viewed in HTML format. The complete attack schedule for the two weeks of testing data can be found in Appendix A of this document.

12.2 Numbers of Attacks

Table 12-1 shows the number of instances of each type of attack that was inserted in the 1998 DARPA intrusion detection evaluation. Each row represents a type of attack. These attack types have been grouped by their category: User to Root, Remote to Local User, Denial of Service, or Probe. The second major column lists the number of instances of the current attack type that were performed in the clear. Within this second column the data is further divided into subcolumns listing the number of attacks of each type in the training data (first seven weeks), the testing data (last two weeks) and the total over all nine weeks of data. The third column shows the number of instances of each attack type that were performed stealthily. Note that no Remote to Local User attacks or Denial of Service attacks were stealthy. Finally, the third column aggregates the sums from the previous two columns, and shows the total number of instances (both clear and stealthy) of attacks of each type. For example, there were 46 Eject attacks in the simulation. Of these 46 Eject attacks, 10 were stealthy and 36 were performed in the clear. Of the 36 clear Eject attack instances, 29 were in the training data, and 7 were in the testing data.

Name	Attacks Performed in the Clear			Stealthy Attacks			All Attacks (Clear + Stealthy)
	Training	Testing	Total	Training	Testing	Total	Total
User to Root	67	24	91	18	5	23	114
Eject	29	7	36	8	2	10	46
Fdformat	12	7	19	4	3	7	26
Fibconfig	11	2	13	6	0	6	19
Perl	15	1	16	0	0	0	16
Ps	0	4	4	0	0	0	4
Xterm	0	3	3	0	0	0	3
Remote to Local User	19	15	34	0	0	0	34
Dictionary	3	2	5	0	0	0	5
Ftp-Write	2	1	3	0	0	0	3
Guest	4	0	4	0	0	0	4
Imap	4	1	5	0	0	0	5
Named	0	3	3	0	0	0	3
Phf	6	1	7	0	0	0	7
Sendmail	0	3	3	0	0	0	3
Xlock	0	2	2	0	0	0	2
Xsnoop	0	2	2	0	0	0	2
Denial Of Service	65	34	99	0	0	0	99
Apache2	0	3	3	0	0	0	3
Back	4	2	6	0	0	0	6
Land	7	2	9	0	0	0	9
Mailbomb	0	1	1	0	0	0	1
Neptune	13	7	20	0	0	0	20
Process Table	0	3	3	0	0	0	2
Smurf	19	8	27	0	0	0	27
Syslog	4	2	6	0	0	0	6
Teardrop	18	4	22	0	0	0	22
UDPS Storm	0	2	2	0	0	0	2
Probe/Surveillance	50	14	64	0	0	0	64
IPSweep	22	3	25	0	0	0	25
Mscan	0	1	1	0	0	0	1
Nmap	12	6	18	5	3	0	18
Saint	0	2	2	0	0	0	2
Satan	16	2	18	0	0	0	18
Total for All Attacks	201	87	288	18	5	23	311

Table 12-1: Instances of Non-Scenario Attacks

12.3 Verification of Attack Success

A major unanticipated difficulty of the process of creating the evaluation dataset was verifying that the inserted attacks actually ran and had the desired effect. Because the simulation network was not designed with automation of this task in mind, all verification of attack success was performed by a human operator after the data had been collected.

The verification process involved looking at logs generated by the "exs" script for each attack after it had run. This process was not ideal. In a more carefully designed verification strategy, the success or failure of an attack instance would be determined automatically at run time by looking at the actual data collected for that attack (tcpdump data, and BSM logs). Without such an automated system in place, the task of verifying attack success turned out to be quite time consuming and prone to error. A system for automatically validating attack success is being developed for use in the 1999 version of the Lincoln Laboratory DARPA intrusion detection evaluation.

Chapter 13

Results and Future Work

13.1 Results of the 1998 Evaluation

The 1998 DARPA intrusion detection evaluation was extremely successful in providing the first comprehensive realistic evaluation of intrusion detection systems. The results were useful in focusing research and highlighting the current capabilities and recent advances of existing intrusion detection systems. The procedures used to create the data and generate attacks worked well and can easily be reused to generate more data for future evaluations. Researchers who participated in the evaluation were able to see the strong and weak points of their own approaches, and the whole intrusion detection research community gained a great deal of insight about the strengths and weaknesses of current efforts in the field as a whole.

Although full discussion of the detailed results of the evaluation are outside of the scope of this thesis, there were a few broad observations that will be useful in guiding future efforts to evaluate intrusion detection systems using simulated computer attacks.

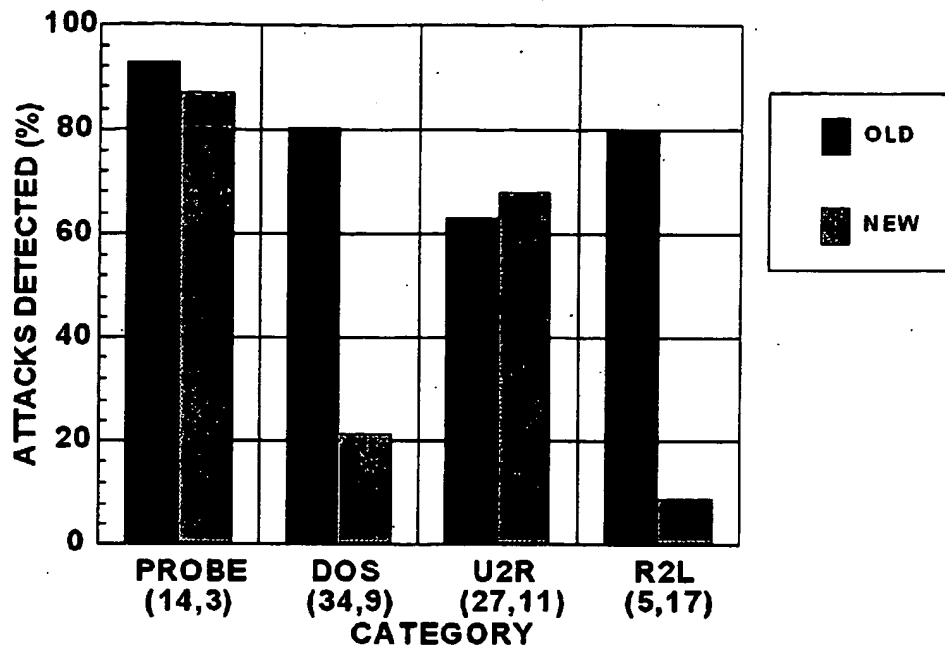


Figure 13-1: Current Intrusion Detection Systems Cannot Find New DoS and Remote to User Attacks

First, current intrusion detection research systems represent a dramatic improvement over older keyword spotting systems. The best combination of 1998 evaluation systems provides more than two orders of magnitude of reduction in false alarm rate with greatly improved detection accuracy. Second, current Intrusion Detection Systems can successfully identify older, known attacks with a low false alarm rate, but do not perform as well when identifying novel or new attacks. Figure 13-1 graphically shows this result by comparing the detection rates of the three best evaluated systems that used tcpdump data. The vertical axis of the chart represents the percentage of attacks detected when set to a threshold that allows for 10 false alarms/day. Results are broken up into the four attack categories (Probe, Denial of Service, User to Root, and Remote to Local) along the horizontal axis. This figure shows us that systems were able to

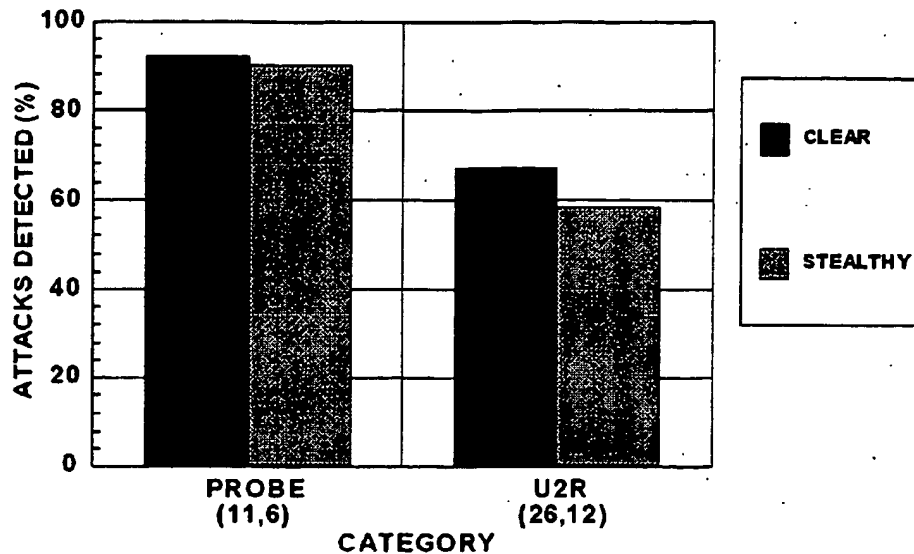


Figure 13-2: Clear vs. Stealthy Attack Detection Rates
 (Top 3 Systems, TCPDump Data, 10 false alarms/day)

generalize well to detect new Probe and User to Root attacks, but almost completely missed new Denial of Service and Remote to Local attacks.

Finally, the stealthy approaches presented in Chapter 11 did not significantly degrade the performance of the evaluated systems. Figure 13-3 compares the detection rate (for the top three systems at a false alarm rate of 10/day) for clear and stealthy Probes and User to Root Attacks. The rate of detection is represented on the vertical axis and Results are broken up into the two attack categories that can be made stealthy (Probes and User to Root) along the horizontal axis. Detection rates for stealthy attacks were not significantly lower than for attacks performed in the clear. Two possible reasons for the relative ineffectiveness of the stealthy attacks are the fact that many stealthy attacks were included in the training dataset, and the fact that many of the stealthy approaches discussed in Chapter 11 were designed specifically to thwart systems

that rely on keyword spotting. More detailed descriptions of these results can be found at the Lincoln Laboratory Intrusion Detection Evaluation Website [42].

13.2 Future Work

13.2.1 Inclusion of Windows NT and Other Systems in the Evaluation

The 1998 Intrusion Detection evaluation focused on attacks targeted at UNIX servers, because at the current time these UNIX based machines are the most common server platform within the defense community. However, use of Windows NT servers is growing in both military and commercial environments, and attacks against Windows NT servers should be included in future evaluations. As the landscape of computer networks continues to evolve, the evaluation must evolve and adapt in order to maintain its relevance and usefulness.

13.2.2 Better Automation of Attack Planning and Verification

In the 1998 evaluation, all attack planning and verification was performed by hand. These tasks proved to be very time intensive and hindered the timely completion of the evaluation. In the future, this process should be automated. All that should be required of the human operator is to supply a database of attacks and their variants. Once this database has been collected, an automated system should be able to plan out a schedule, and generate the "exs" scripts for all of the attack instances in the evaluation. After the "exs" scripts have been run, automatic verification routines should be developed to look at the collected data and validate that the attacks occurred successfully. Developing these

automated tools will take a great deal of effort, but once they have been completed the process of conducting future evaluations will be greatly simplified.

13.2.3 Improved Stealthy Attacks

The results presented in section 13.1 show that the stealthy methods used in the 1998 DARPA evaluation were largely unsuccessful. Stealthy attacks were ineffective because (1) Examples of stealthy attacks were provided in the training data and many intrusion detection systems were trained to find stealthy actions instead of the actual break-in, (2) Only a small part of the development effort focussed on making attacks stealthy, and (3) All of the user to root attacks created a root shell, which makes them relatively easy to detect. Future evaluation efforts should dedicate a large percentage of total attack development time to further developing and exploring new methods of achieving stealthiness.

Appendix A

Attack Schedule for Test Dataset

Day	Name	Time	Source	Dest	Type	Stealthy
Week 1						
Mon	portsweep	10:51:21	207.136.86.223	marx	probe	
Mon	pod	14:49:52	207.103.80.104	marx	dos	
Mon	teardrop	16:08:16	123.123.123.123	zeno	dos	
Mon	syslog	16:46:28	1.1.1.1	pascal	dos	
Tues	satan	08:31:46	153.107.252.61	marx	probe	
Tues	format	09:11:57	209.74.60.168	pascal	u2r	Y
Tues	apache2	10:11:30	207.181.92.211	marx	dos	
Tues	format2	15:44:55	194.27.251.21	pascal	u2r	
Tues	neptune	18:15:57	166.102.114.43	zeno	dos	
Tues	smurf	19:43:50	100subs	marx	dos	
Wed	httptunnel	09:32:05	197.182.91.233	pascal	scenario	
Wed	nmap	10:34:10	207.253.84.13	linux2	probe	
Wed	teardrop	16:30:05	123.123.123.123	zeno	dos	
Wed	smurf	20:07:38	199.174.194	marx	dos	
Wed	eject	21:52:01	153.10.8.174	pascal	u2r	Y
Thurs	pod	10:16:10	152.204.242.193	*	dos	
Thurs	nmap	10:23:19	207.253.84.13	linux2	probe	
Thurs	named	10:33:08	152.169.215.104	linux	r2l	
Thurs	eject	14:32:32	194.7.248.153	pascal	u2r	Y
Thurs	land	14:45:13	zeno	zeno	dos	
Thurs	smurf	15:10:48	10subs	linux3	dos	
Fri	snmpgetattack	08:03:06	207.230.54.203	loud	scenario	
Fri	teardrop	09:27:09	194.7.248.153	marx	dos	
Fri	format	10:06:37	194.7.248.153	pascal	u2r	Y
Fri	back	12:41:54	204.97.153.43	marx	dos	
Fri	neptune	14:41:28	9.9.9.9	pascal	dos	
Fri	processtable	19:20:35	Calvin	pascal	dos	
Fri	neptune	19:50:59	10.20.30.40	pascal	dos	
Week 2						
Mon	snmpgetattack	08:03:07	207.230.54.203	loud	scenario	
Mon	named	08:55:02	calvin	marx	r2l	
Mon	xlock	09:11:04	194.7.248.153	marx	r2l	
Mon	xlock2	09:27:27	194.7.248.153	pascal	R2l	
Mon	smurf	09:30:41	10subs	pascal	dos	
Mon	multihop	09:36:10	206.229.221.82	marx	scenario	
Mon	ipsweep	09:36:07	204.97.153.43	*	probe	
Mon	xsnmp	09:38:58	194.27.251.21	pascal	r2l	
Mon	named	09:54:39	204.97.153.43	linux1	r2l	
Mon	smurf	10:34:11	all.attackers	marx	dos	
Mon	saint	11:01:23	calvin	pascal	probe	
Mon	ffb	11:15:16	197.218.177.69	pascal	u2r	
Mon	portsweep	14:01:26	202.247.224.89	pascal	probe	
Mon	pod	16:11:36	207.103.80.104	pascal	dos	
Mon	apache2	18:22:34	196.227.33.189	marx	dos	
Mon	format	20:46:48	195.73.151.50	pascal	u2r	
Mon	udpstorm	21:06:52	172.16.112.50	Zeno	dos	
Mon	format2	23:10:31	202.49.244.10	Pascal	u2r	
Mon	smurf	23:16:43	152.169.215.202	Marx	dos	

			.77.162			
Mon	smurf	23:26:37	255.255.255.255 .et.al	Pascal	dos	
Tues	warezmaster	08:08:29	200.27.121.118	Pascal	scenario	
Tues	portsweep	08:27:28	135.8.60.182	Marx	dos	
Tues	multihop	09:00:19	206.229.221.82	Marx	scenario	
Tues	perlmagic	09:08:32	166.102.114.43	Marx	u2r	
Tues	xsnop	09:09:03	200.27.121.118	Pascal	r2l	
Tues	pod	10:33:08	209.30.70.14	Linux1	dos	
Tues	dict1	11:23:37	208.239.5.230	Marx	r2l	
Tues	format1	11:55:17	209.30.71.165	Pascal	u2r	Y
Tues	sendmail	12:14:12	204.97.153.43	Marx	r2l	
Tues	eject	13:20:35	195.115.218.108	Pascal	u2r	
Tues	satan	15:17:26	208.253.77.185	Zeno	probe	
Tues	mscan	17:36:25	207.75.239.115	*	probe	
Tues	processtable	18:04:07	calvin	Pascal	Dos	
Tues	smurf	21:03:36	209.1.12.*	Linux2*	dos	
Wed	snmpgetattack	08:03:06	207.230.54.203	Loud	scenario	
Wed	warezclient	08:15:06	all.attackers	Pascal	scenario	
Wed	ps	10:06:44	209.154.98.104	Pascal	u2r	
Wed	nmap	10:16:15	207.253.84.13	Linux2	probe	
Wed	dict1	10:24:39	152.169.215.104	Marx	r2l	
Wed	xterm	10:26:12	194.27.251.21	Linux1	u2r	
Wed	rootkit	10:33:27	153.10.8.174	Marx	scenario	
Wed	xterm	10:52:58	197.182.91.233	Linux1	r2l	
Wed	neptune	11:11:28	10.140.175.80	Zeno	dos	
Wed	udpstorm	12:40:23	pascal	Zeno	dos	
Wed	loadmodule	13:32:38	205.180.112.36	Zeno	u2r	
Wed	syslog	13:46:54	194.7.248.153	Pascal	dos	
Wed	imap	14:36:15	calvin	Marx	r2l	
Wed	back	15:46:55	209.117.157.183	Marx	dos	
Wed	ps	19:26:57	197.218.177.69	Pascal	u2r	
Wed	ipsweep	20:07:16	194.7.248.153	*	probe	
Wed	ffb	20:59:25	208.239.5.230	Pascal	u2r	
Thurs	snmpgetattack	08:03:06	207.230.54.203	Loud	Scenario	
Thurs	warezclient	08:15:05	all.attackers	Pascal	Scenario	
Thurs	multihop	09:00:09	206.229.221.82	Pascal	Scenario	
Thurs	eject	09:15:00	135.008.060.182	Pascal	u2r	
Thurs	portsweep	09:36:05	195.73.151.50	Pascal	Probe	Y
Thurs	nmap	09:41:52	207.253.84.13	Linux2	probe	
Thurs	sendmail	10:44:06	207.230.54.203	Marx	R2l	
Thurs	loadmodule	11:37:27	135.13.216.191	Zeno	U2r	
Thurs	eject	11:59:43	197.182.91.233	Pascal	U2r	
Thurs	httptunnel	12:30:03	pascal	Calvin	scenario	
Thurs	neptune	13:30:49	209.74.60.168	Zeno	Dos	
Thurs	eject2	13:42:55	197.182.91.233	Pascal	U2r	
Thurs	processtable	17:56:34	calvin	Pascal	Dos	
Thurs	neptune	20:15:14	135.13.216.191	Pascal	Dos	
Thurs	format	21:13:12	206.186.80.111	Pascal	U2r	
Thurs	mailbomb	21:14:46	204.233.47.21	Pascal	Dos	
Thurs	nmap	23:26:16	128.223.199.68	Zeno	Probe	
Fri	snmpgetattack	08:04:11	207.230.54.203	Loud	Scenario	
Fri	warezclient	08:16:11	all.attackers	Pascal	Scenario	
Fri	xterm	08:43:22	calvin, 209.167. 99.71	Linux1	u2r	
Fri	ftp-write	08:49:26	194.27.251.21	Pascal	R2l	
Fri	ps	08:53:05	207.136.86.223	Pascal	U2r	
Fri	httptunnel1	09:01:08	pascal	Marx	scenario	
Fri	teardrop	09:33:40	222.222.222.222	Marx	Dos	
Fri	saint	10:20:15	197.218.177.69	*	probe	

Fri	neptune	10:30:11	18.28.38.48	Pascal	Dos	
Fri	ps2	12:27:21	202.49.244.10	Pascal	u2r	
Fri	eject	12:48:24	153.107.252.61	Pascal	u2r	
Fri	land	13:27:39	Zeno	Zeno	u2r	
Fri	sendmail	16:22:02	194.7.248.153	Marx	R2l	
Fri	ipsweep	16:46:15	135.13.216.191	*	Probe	
Fri	pod	18:16:42	196.227.33.189	linux1	Dos	
Fri	phf	20:18:38	197.182.91.233	Marx	r2l	
Fri	apache2	21:04:50	202.72.1.77	Marx	dos	

References

- [1] Anderson, D., T. Lunt, H. Javitz, A. Tamaru, and A. Valdes. "Safeguard Final Report: Detecting Unusual Program Behavior Using the NIDES Statistical Component," Computer Science Laboratory, SRI International, Menlo Park, CA, Technical Report, December 1993.
- [2] Anonymous. *Maximum Security: A Hacker's Guide to Protecting Your Internet Site and Network*, Chapter 15, pp. 359-362. Sams.net, 201 West 103rd Street, Indianapolis, IN, 46290. 1997.
- [3] Bishop, M., S. Cheung, et al. "The Threat from the Net", IEEE Spectrum, 38(8). 1997.
- [4] Bugtraq Archives (e-mail regarding Apache vulnerability). http://www.geek-girl.com/bugtraq/1998_3/0442.html. August 7, 1998.
- [5] Bugtraq Archives. http://geek-girl.com/bugtraq/1997_4/0283.html. November 13, 1999.
- [6] Bugtraq Archives. http://www.geek-girl.com/bugtraq/1999_1/0852.html. February 19, 1999.
- [7] CERT Advisory CA-93.10. http://www.cert.org/ftp/cert_advisories/CA-93%3a10.anonymous.FTP.activity. July 14, 1993.
- [8] CERT Advisory CA-93.18,CA-95:12. http://www.cert.org/ftp/cert_advisories/CA-95:12.sun.loadmodule.vul. September 19, 1997.
- [9] CERT Advisory CA-95.09. http://www.cert.org/ftp/cert_advisories/CA-95%3a09.Solaris-ps.vul. August 20, 1995.
- [10] CERT Advisory CA-96.01. http://www.cert.org/ftp/cert_advisories/CA-96.01.UDP_service_denial. February 8, 1996.
- [11] CERT Advisory CA-96.06. http://www.cert.org/ftp/cert_advisories/CA-96.06.cgi_example_code. March 20, 1996.
- [12] CERT Advisory CA-96.12. http://www.cert.org/ftp/cert_advisories/CA-96.12.suidperl_vul. June 26, 1996.
- [13] CERT Advisory CA-96.21. http://www.cert.org/ftp/cert_advisories/CA-96.21.tcp_syn_flooding. September 19, 1996.
- [14] CERT Advisory CA-96.26. http://www.cert.org/ftp/cert_advisories/CA-96.26.ping. December 16, 1996.
- [15] CERT Advisory CA-97.05. http://www.cert.org/ftp/cert_advisories/CA-97.05.sendmail. January 28, 1997.
- [16] CERT Advisory CA-97.09. http://www.cert.org/ftp/cert_advisories/CA-97.09.imap_pop. April 7, 1997.
- [17] CERT Advisory CA-97.28. http://www.cert.org/ftp/cert_advisories/CA-97.28.Teardrop_Land. December 16, 1997.
- [18] CERT Advisory CA-98.01. http://www.cert.org/ftp/cert_advisories/CA-98.01.smurf. January 5, 1998.

- [19] CERT Advisory CA-98.05. http://www.cert.org/ftp/cert_advisories/CA-98.05.bind_problems. April 8, 1998.
- [20] CERT Incident Note. http://www.cert.org/incident_notes/IN-98.02.html. July 2, 1998.
- [21] CERT Vulnerability Note VN-98.01. http://www.cert.org/vul_notes/VN-98.01.XFree86.html. May 3, 1998.
- [22] Computer Emergency Response Team Website. <http://www.cert.org>.
- [23] Cisco Systems, Inc. "NetRanger Intrusion Detection System Technical Overview," http://www.cisco.com/warp/public/778/security/netranger/ntran_tc.htm
- [24] COAST FTP Site. <ftp://coast.cs.purdue.edu/pub/tools/unix/satan/>. 1998
- [25] Cunningham, R. K., R. P. Lippmann, D. J. Fried, S.L. Garfinkel, I. Graf, K. R. Kendall, S.E. Webster, D. Wyschogrod, and M. A. Zissman, (1999) "Evaluating Intrusion Detection Systems without Attacking your Friends: The 1998 DARPA Intrusion Detection Evaluation," In Proceedings ID'99, Third Conference and Workshop on Intrusion Detection and Response, San Diego, CA: SANS Institute.
- [26] S. Durst, T. Champion. Packet Address Swapping for Network Simulation. Patent application, Air Force Research Laboratory. March 1999.
- [27] Simson Garfinkel and Gene Spafford. *Practical Unix & Internet Security*. O'Reilly & Associates, Inc., 101 Morris Street, Sebastopol CA, 95472, 2nd edition, April 1996.
- [28] Heberlein, T. "Network Security Monitor (NSM) – Final Report", U.C. Davis: February 1995, <http://seclab.cs.ucdavis.edu/papers/NSM-final.pdf>.
- [29] L. Todd Heiberlein, Gihan V. Dias, Karl N. Levit, Biswanath Mukherjee, Jeff Wood, and David Wolber. A network security monitor. In *Proceedings of the 1990 Symposium on Research in Security and Privacy*, pages 296-304, Oakland, CA, May 1990. IEEE.
- [30] Heberlein, T.L., G.V. Dias, K.N. Levitt, B. Mukherjee, J. Wood, and D. Wolber. "A Network Security Monitor", in 1990 IEEE Symposium on Research in Security and Privacy. pp. 296-304.
- [31] *hobbit* hobbit@avian.org. Netcat README file. <http://www.l0pht.com/~weid/netcat/readme.html>. 1998.
- [32] D. Icove, K. Seger, W. VonStorch. *Computer Crime: A Crimefighter's Handbook*. O'Reilly & Associates, Inc., 101 Morris Street, Sebastopol CA, 95472, August 1995.
- [33] Koral Ilgun. USTAT : A real-time intrusion detection system for UNIX. Master's thesis, University of California Santa Barbara, November 1992.
- [34] Impack binaries on Rootshell.com. <http://www.rootshell.com/archive-j457nxiqi3gq59dw/199804/impack103.tar.gz.html>. April 13, 1998.
- [35] Internet Security Systems X-Force. <http://www.iss.net>.
- [36] R. Jagannathan, Teresa Lunt, Debra Anderson, Chris Dodd, Fred Gilham, Caveh Jalai, Hal Havitz, Peter Neumann, Ann Tamaru, and Alfonso Valdes. System design document: Next-generation intrusion detection expert system (NIDES). Technical report, Computer Science Laboratory, SRI International, Menlo Park, CA 94025, March 1993.

- [37] Javitz, H.S. and A. Valdes, "The NIDES Statistical Component Description and Justification," Computer Science Laboratory, SRI International, Menlo Park, CA, Technical Report, March 1994.
- [38] Kemmerer, Rachard A. "NSTAT: A Model-based Real-time Network Intrusion Detection System," Computer Science Department, University of California, Santa Barbara, Report TRCS97-18, <http://www.cs.ucsb.edu/TRs/TRCS97-18.html>
- [39] Ko, C., M. Ruschitzka, and K. Levitt. "Execution Monitoring of Security-Critical Programs in a Distributed System: A Specifications-Based Approach," In Proceedings 1997 IEEE Symposium on Security and Privacy, pp. 134-144, Oakland, CA: IEEE Computer Society Press.
- [40] Lawrence Berkeley National Laboratory, Network Research Group Homepage. <http://www-nrg.ee.lbl.gov/>. May 1999.
- [41] Lawrence Livermore National Laboratory. "Network Intrusion Detector (NID) Overview," Computer Security Technology Center, <http://ciac.llnl.gov/cstc/nid/intro.html>.
- [42] Lincoln Laboratory ID Evaluation Website, MIT, <http://www.ll.mit.edu/IST/ideval/index.html>. 1999.
- [43] Lippmann, R.P., Cunningham, R.K., Webster, S.E., Graf, I., Fried, D. Using Bottleneck Verification to Find Novel New Computer Attacks with a Low False Alarm Rate. Unpublished Technical Report. 1999.
- [44] Lunt, T.F. "Automated Audit Trail Analysis and Intrusion Detection: A Survey", in Proceedings 11th National Computer Security Conference., pp. 65-73. 1988.
- [45] NMAP Homepage. <http://www.insecure.org/nmap/index.html>. 1998
- [46] Parris, Phillip A. and Peter G. Neumann. "EMERALD: Event Monitoring Enabling Response to Anomalous Live Disturbances," In Proceedings 20th National Information Systems Security Conference, Oct 7, 1997.
- [47] Paxon, Vern. "Bro: A System for Detecting Network Intruders in Real-Time," In Proceedings of the 7th USENIX Security Symposium, San Antonio, TX, January 1998, <http://www.aciri.org/vern/paper.html>.
- [48] Nicholas Puketza, Kui Zhang, Mandy Chung, Biswanath Mukherjee, Ronald Olsson. A Methodology for Testing Intrusion Detection Systems. Technical report, University of California, Davis, Department of Computer Science, Davis, CA 95616, September 1995.
- [49] *Real Secure 2.5 User Manual*, Chapter 6. Internet Security Systems, Atlanta, GA. [available at <http://download.iss.net/manuals/rs25.tar.Z>]
- [50] Rootshell Website. <http://www.rootshell.com>. 1999.
- [51] Rootshell Website. <http://www.rootshell.com/archive-j457nxiqi3gq59dv/199707/perl-ex.sh.html>. August 26, 1996.
- [52] Rootshell Website. http://www.rootshell.com/archive-j457nxiqi3gq59dv/199707/solaris_ps.txt.html. June 11, 1997.
- [53] Rootshell Website. <http://www.rootshell.com/archive-j457nxiqi3gq59dv/199707/fdformat-ex.c.html>. March 24, 1997.
- [54] Rootshell Website. http://www.rootshell.com/archive-j457nxiqi3gq59dv/199711/sol_syslog.txt.html. November 6, 1997.

- [55] Rootshell Website. <http://www.rootshell.com/archive-j457nxiqi3gq59dv/199801/beck.tar.gz.html>. Jan 1, 1998.
- [56] Rootshell Website. http://www.rootshell.com/archive-j457nxiqi3gq59dv/199805/xterm_exp.c.html. June 4, 1998.
- [57] Rootshell Website. <http://www.rootshell.com/archive-j457nxiqi3gq59dv/199806/mscan.tgz.html>. June 24, 1998.
- [58] Saint Website. <http://www.wvdsi.com/saint>. 1998.
- [59] S. Staniford-Chen, S. Cheung, R. Crawford, M. Dilger, J. Frank, K. Levitt, C. Wee, R. Yip, D. Zerkle, and J. Hoagland. GrIDS—a graph based intrusion detection system for large networks. 1996. [available at <http://seclab.cs.ucdavis.edu/arpa/grids/welcome.html>]
- [60] Sun Microsystems Security Bulletin: #00138. <http://sunsolve.Sun.com/pub-cgi/us/sec2html?secbull/138>. 17 April, 1997.
- [61] Sun Microsystems Security Bulletin: #00140. <http://sunsolve.Sun.com/pub-cgi/us/sec2html?secbull/140>. 14 May, 1997.
- [62] Sun Microsystems, Solaris Security Website. <http://www.sun.com/solaris/2.6/ds-security.html>. May 1999.
- [63] Sundaram, A. "In Introduction to Intrusion Detection", Crossroads: The ACM Student Magazine, 2(4). 1996.
- [64] Daniel Weber. A Taxonomy of Computer Intrusions. Master's thesis, Massachusetts Institute of Technology, Cambridge, MA, 02139, 1998.
- [65] Seth Webster. The Development and Analysis of Intrusion Detection Algorithms. Master's Thesis, Massachusetts Institute of Technology, Cambridge, MA, 02139, 1998.

(D) Art submitted by Applicant - Entered

The following items (1) – (7) listed below are hereby entered as evidence submitted to the Examiner as noted on an IDS transmission received by the USPTO on September 09, 2002.

- (1) Copy of EP Patent Number 1081918 ("Hinde et al."). This evidence was entered into the record by the Applicant on 09-09-2002.
- (2) Copy of EP Patent Number 0991244 ("Chapman et al."). This evidence was entered into the record by the Applicant on 09-09-2002.
- (3) Copy of WIPO Patent Number 200131852 ("Burnett et al."). This evidence was entered into the record by the Applicant on 09-09-2002.
- (4) Copy of WIPO Patent Number 200067443 ("Nilsen"). This evidence was entered into the record by the Applicant on 09-09-2002.
- (5) Copy of WIPO Patent Number 200028431 ("Jacobs et al."). This evidence was entered into the record by the Applicant on 09-09-2002.
- (6) Copy of International Search Report of International Patent Number PCT/US 01/47067. This evidence was entered into the record by the Applicant on 09-09-2002.
- (7) Copy of NPL- Self-Help for Bugs in the Field, Circuit Cellular, The Magazine for Computer Applications XP-002208351. This evidence was entered into the record by the Applicant on 09-09-2002.

Copies of all References follows.

//

(12) **EUROPEAN PATENT APPLICATION**

(43) Date of publication:
07.03.2001 Bulletin 2001/10

(51) Int Cl.7: **H04L 29/06**

(21) Application number: **00307357.4**

(22) Date of filing: **25.08.2000**

(84) Designated Contracting States:
AT BE CH CY DE DK ES FI FR GB GR IE IT LI LU
MC NL PT SE
 Designated Extension States:
AL LT LV MK RO SI

• **Wilcock, Lawrence**
Malmesbury, Wiltshire SN16 9TV (GB)
 • **Low, Colin**
Wotton-U-Edge, Gloucestershire GL12 7LT (GB)

(30) Priority: **04.09.1999 GB 9920834**

(71) Applicant: **Hewlett-Packard Company**
Palo Alto, California 94304-1112 (US)

(74) Representative:
Lawrence, Richard Anthony et al
Hewlett-Packard Limited,
IP Section,
Building 3,
Filton Road
Stoke Gifford, Bristol BS34 8QZ (GB)

(72) Inventors:
 • **Hinde, Stephen John**
Redland Bristol BS6 7DH (GB)

(54) **Providing secure access through network firewalls**

(57) To enable controlled, secure connections using a versatile protocol such as TCP/IP to be established through a firewall and proxy server, the versatile protocol is tunnelled using HTTP. Client to server communications are effected using an HTTP POST operation. Server to client communications are effected using an HTTP GET operation to establish a tunnelled socket;

this socket is closed within an interval less than any timeout imposed by the proxy server and immediately re-established by another GET operation, irrespective of whether data continue to be pending for communication to the client. A globally-unique ID is included in each POST and GET message to enable related messages to be recognized.

Client ← Service

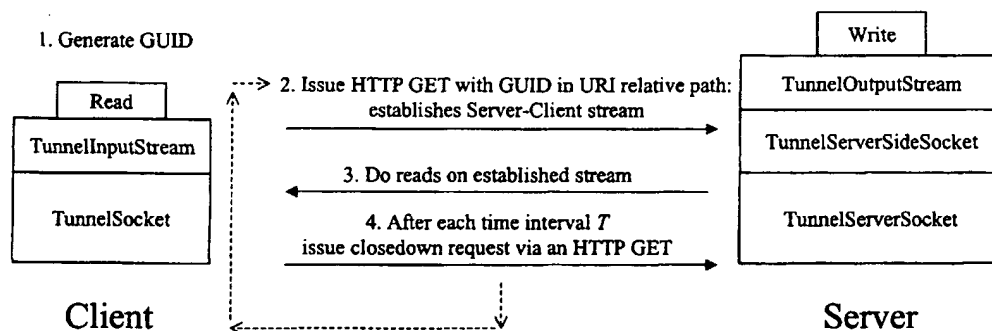


Fig.5

DescriptionTechnical Field

- 5 [0001] This invention relates to methods and apparatus for providing secure access between a service external to a network firewall (such as a web page server) and a client internal to the firewall (such as a web browser).

Background Art

- 10 [0002] Facilities for global access to information, such as the World Wide Web, are undergoing immense growth and expansion, both in volume of data transferred and in sophistication of tools, such as web browsers, for accessing those data. This in turn requires the flexibility and capability of the data network infrastructure to be increased, without at the same time jeopardising its security features. Often these objectives conflict.

- 15 [0003] Web based applications are "client-server" in nature. A client-based browser runs on an end user's computer, communicating via network links with a web server operated by a service provider. The server provides data defining the content of web pages which are rendered and displayed by the user's browser; typically the web page content is defined using a markup language such as Hypertext Markup Language (HTML). The communications between the browser and the web server are conducted in accordance with a protocol called Hypertext Transfer Protocol (HTTP), defined in the Internet Engineering Task Force's RFC 1945. HTTP is a simple text-based protocol that has the peculiar quality that messages conforming to it are trusted and allowed to pass around the internets, intranets and extranets that make up today's "Internet". This Internet is really a series of closely coupled networks linked together through "firewalls": network nodes that allow controlled, restricted access between two networks. However the ability to "browse the world wide web" is seen as a universal common denominator, and as such HTTP messages are allowed to pass through these firewalls unchecked. There have been a number of enhancements to the HTTP protocol: the description
20 herein relates by way of example to the basic version, HTTP/1.0, which is supported universally and with which later versions of HTTP are backwards compatible. Nonetheless the invention is not limited to use with HTTP/1.0 or indeed any other specific version of HTTP, and the claims hereof should be construed accordingly.

- [0004] Web pages defined using the markup language can be enhanced by the use of code written in the Java programming language; this allows dynamic content and interactivity to be added to web pages. Java components can run either on the server end of the network connection, as "servlets", or on the client browser machine, in which case they are known as "applets". Many potential security problems were envisaged with the introduction of applets, such as "viruses" and "trojan horses", so a tight set of security restrictions were imposed on what applets could and could not do. To this end applets are executed on the client machine in a controlled environment called a "sandbox". This sandbox defines how the applet can interact with the resources available in the computational platform the applet is running on, via a limited application programming interface (API). For example, the applet typically cannot interact with the local disk, nor connect to other computers on the network in unrestricted fashion. However the applet can typically connect back to the web server it was served from, although by way of an HTTP connection only. References to Java herein relate by way of example to Java/1.1 which is widely deployed; later versions of Java provide enhanced network support, but are backwards compatible with this base version.

- 40 [0005] The first generation of web content was mainly very static in nature - like pages from a magazine with text and pictures. The second generation of web content became increasingly dynamic, providing a user interface for applications, such as database queries. The third generation of web content is becoming increasingly interactive, with real-time communications, such as video, text chat and Internet Telephony, being added as an integral part. Internet-based client-server applications normally operate by opening "sockets" between the client and server, using Transaction Control Protocol/Internet Protocol (TCP/IP). TCP/IP sockets provide bi-directional, reliable communications paths. These can be used to implement dynamic or interactive client-server based applications, such as are required by the second and third generations of web content.

- [0006] However, these more sophisticated applications pose a problem, in that the communications protocols they often require for interaction with the web server, such as TCP/IP, are intentionally barred by firewalls and proxy servers.
50 It is therefore an object of this invention to provide clients, such as Java applets in a sandbox, with some controlled ability (within the context of a specific web-based application) to interact through a firewall with other resources, such as web servers, using protocols in addition to HTTP.

Disclosure of Invention

- 55 [0007] According to one aspect of this invention there is provided a method of permitting secure access between a service external to a network firewall and a client internal to the firewall, comprising the steps of:

- (a) effecting an HTTP GET operation or equivalent thereof (which may be in a protocol functionally equivalent to HTTP) from the client to establish a communications socket for communicating data from the service to the client;
- (b) after a predetermined interval effecting another GET operation or equivalent thereof to close the communications socket, irrespective of whether access between the service and the client is required to continue; and
- repeating steps (a) and (b) while access between the service and the client is required to continue.

[0008] The HTTP Tunneling Socket described herein as an example of this invention is a socket that operates on top of the HTTP protocol. For interaction with a Java-based application it provides the standard APIs as defined for a socket in the Java/1.1 API specification. For interaction with the network it uses the HTTP/1.0 protocol. This technique in which one protocol is carried in messages conveyed over another protocol is known as tunneling. The use of tunneling has two benefits. It allows the tunnel to be executed in the browser sandbox, since the APIs for accessing an HTTP connection to a web server do not have any security restrictions. Secondly it allows the client-server connection to traverse firewalls between intranets and internets, as HTTP is the protocol that all firewalls allow to pass through them.

[0009] The Java applet and the server are thus provided with a connection protocol (such as TCP/IP) which is more versatile and capable than the HTTP connection to which they would normally be limited by the firewall and/or proxy server. However, the restrictions imposed by the sandbox prevent malicious or inappropriate access to the client machine's resources through the tunnelled protocol.

Brief Description of Drawings

[0010] A method and apparatus in accordance with this invention, for permitting secure access between a client behind a firewall and a service provided by a node (such as a server) external to the firewall, will now be described, by way of example, with reference to the accompanying drawings, in which:

- Figure 1 is a schematic illustration of circumstances in which the invention can be used;
- Figure 2 shows an outline of an implementation of the invention in the Java programming language;
- Figure 3 illustrates the basic principles of operation of the invention;
- Figure 4 shows in more detail the mechanism for communications from the client to the server; and
- Figure 5 shows in more detail the mechanism for communications from the server to the client.

Best Mode for Carrying Out the Invention, & Industrial Applicability

[0011] There are in principle two aspects involving firewalls to be considered when establishing a connection between an applet in a user's terminal and a web server. The first aspect is the case of a firewall-protected web server to which a browser running on a machine outside the firewall is trying to connect. This aspect does not present a serious problem since the web-server owner can configure the firewall to allow this connection to be made.

[0012] The second aspect is the case of a web browser machine in a corporate intranet trying to connect through the corporate firewall to a web server on the public Internet, as illustrated in Figure 1. The problem here is that the web server owner has, by design, no control over the firewall. The complete scenario is that the browser 10 running java applet code on the user's terminal will connect initially to a machine 12 called a proxy within the corporate intranet 14. This proxy makes the required connections through the firewall 16 to the internet-based web server 18 on behalf of the browser 10, without exposing sensitive information about the browser and its host terminal to view in the public internet. The proxy 12 also performs various security checks, for example to ensure that the requests are valid HTTP requests.

[0013] However, because of the restriction to the use of HTTP, the web server is unable to provide desirable services to the client, such as real-time audio, video or interactive chat, which are possible with the use of more powerful protocols such as TCP/IP. These more powerful protocols are normally barred from unrestricted use because of the threat they pose of enabling inappropriate or intrusive access to resources within the corporate intranet.

[0014] This problem is overcome by using the invention to establish a "tunnel" through the firewall, carried by the HTTP messages, and thus open communications sockets which the Java applet and the web server can use to communicate with one another. In this way the web server is enabled to use resources on the browser's host machine, albeit still within the tight constraints imposed by the Java sandbox.

[0015] The tunnel consists of two Java programming classes, one (called the TunnelSocket) which implements the "Socket" class as defined in the Java/1.1 specification, and another (called TunnelServerSocket) which implements the "ServerSocket" class defined in that specification. As shown in Figure 2, client-based Java code can access the standard APIs (such as open, read, write, close, etc.) provided by a class conforming to the Socket class definition to communicate with another Java application, such as one running on the web server. The server-based Java application can likewise use the same Socket APIs, plus the standard APIs such as accept etc. provided for servlets by the

ServerSocket class, to communicate with the client-based Java applet. Messages between the Socket class and the ServerSocket class are carried within HTTP messages which traverse the firewall between the client and the server.

[0016] The HTTP tunnel uses the underlying simple text-based HTTP protocol to produce a reliable connection-based protocol between the TunnelSocket and TunnelServerSocket classes. Basic HTTP transactions use one of two methods in the HTTP request message. The GET method is designed to "get" a web page from a web server and the POST method is used to send data to a remote web resource or web application. In implementing the present invention, a GET based protocol is used for the client to server direction, and a POST based protocol is used for the reverse, server to client, direction, as shown in Figure 3. In the event that HTTP is supplanted by a functionally equivalent protocol, it is envisaged that the invention could be used with operations in that protocol equivalent to the HTTP GET and POST operations.

[0017] A single connection between the TunnelSocket and TunnelServerSocket classes is likely to comprise multiple HTTP transactions. To associate these transactions together, a numeric Globally Unique ID (GUID) is generated using a combination of random numbers, network address and time of day, so that each GUID generated is a unique quantity both in relation to time and location in the network. This GUID is incorporated in each HTTP request, both GET and POST, so that the HTTP protocol can recognize that they relate to the same communications socket. Specifically, the GUID is passed as the HTTP Uniform Resource Indicator (URI) (the field used to specify a document name in normal use of HTTP).

[0018] Client to server connection is implemented using the POST HTTP operation. Each time a write is performed to the client socket (TunnelSocket class) the request is packaged up and sent as the payload of an HTTP POST message, including the relevant GUID, as shown in Figure 4. Each such write operation sends a separate POST message through the firewall.

[0019] The server to client direction is more complex because of restrictions imposed by the proxy server 12 and the firewall 16. As shown in Figure 5, the client issues an HTTP GET message which tells the server that a new tunneled socket is being established. The HTTP GET request will generate a temporary server-client socket that allows data to be written from the server to the client. However as a security precaution most proxy server implementations will close down such a server-client socket after a time interval P , typically of the order of minutes in duration. Accordingly the tunnel established in this invention periodically itself forces a pre-emptive close down of the connection at some periodicity T which is shorter in duration than the interval P . This is accomplished by issuing another GET request, to force the TunnelServerSocket class to close the existing socket to the client and immediately establish a new socket. It should be noted that this closure (and subsequent reopening) are performed even though data are still pending to be transferred from the server to the client. This approach avoids the overhead of previous techniques, in which the client polls the server at some preselected polling interval, usually of the order of 20 milliseconds. These frequent polls are equivalent to multiple web page requests, so the server receives a large number of web hits per second, imposing a significant overhead and severely restricting the scalability of such prior techniques to handle large numbers of clients simultaneously.

[0020] An illustration of the implementation of the invention is provided below, in the form of pseudo-code, which emphasizes the significant aspects of the implementation but for the sake of clarity omits minor details which, although required in practice, are well within the capability of a person skilled in the art. Likewise the existence is assumed of certain subsidiary functions, such as guidFactory for creating a GUID and createFifoInputStream for reserving and configuring memory for use as a first-in-first-out (FIFO) buffer; again these functions are individually well known in the art, so their internal details do not need to be specified. Depending on the particular manner of implementation, additional parameters may be included, for example, in some function calls; this is indicated by an ellipsis (...).

1. Server to Client direction.1a. TunnelSocket InputStream Code - runs on client

```

5
//
// Open an Input Stream to the Server
// This is the Server to Client direction of the Tunnel Socket.
//
10
// serverAdresss - Internet address e.g. fred.hpl.hp.com:
// standard socket
// Port number is a port number on the server machine to
// connect to: standard socket
15
public open(serverAddress, portNumber) {

    // Generate a new Globally unique id for the new socket.
    // The id is unique across the whole network and across
    // time.
20
    // It is generated using a random number generator, the
    // node's network address and the current time.
    guid = guidFactory();

    // Create a Fifo stream that is used by the application code
    // to read data.
25
    // The underlying protocol reads data off the wire into this
    // Fifo ready for the application.
    fifoInputStream = createFifoInputStream();

30
    // Issue an HTTP/1.0 GET request to the server
    // The on-the-wire format is:
    // GET <guid>?command=establish
    // Content-Type=application/octet-stream
35
    inputStream = issueHttpGetRequest(uri=guid,
        queryString="command=establish",
        Content-Type="application/octet-stream",
        address=serverAddress, port=port);

40
    // Start a timer thread that is invoked at a periodicity of
    // GETTimeOut seconds.
    // Typically GETTimeOut is a few minutes in duration - but
    // it must be less than the time out period of the proxy.
    // The function switchGETStream() is called at the
45
    // periodicity with its own thread of execution.
    startPeriodicTimeOutDemon(switchGETStream(), GETTimeOut);

}

50

// Send a new request to the server telling it to close down
// the current stream and establish a new stream
55
// periodically

```

```

private switchGETStream() {

    // Lock the Input Stream to prevent any race conditions with
    //   readInputStream
    lock(inputStreamLock)

    newInputStream = issueHttpRequest(uri=guid,
    queryStrings="command=switch",
    Content-Type="application/octet-stream",
    address=serverAddress, port=port);

    // Drain the contents of the stream until it is closed by
    //   the remote end
    while (read(InputStream, FifoStream) != EOF);
        writeToFifo(FifoInputStream);

    inputStream = newInputStream();
    unlock(inputStreamLock);

    return();
}

// Read data from the stream via the Fifo - the Fifo buffers
//   data up ready to be read
public readInputStream(..., buf, len) {
    ...
    // if there isn't enough data in the Fifo to satisfy this
    //   read request - top it up from the wire.
    if (!FifoStream.length() > len) {

        //Transfer data from the stream established to the Server
        lock(inputStreamLock);
        writeToFifo(FifoInputStream, read(InputStream,
            len - FifoStream.length()));
        unlock(inputStreamLock);
    }

    // return data in from Fifo
    return(read(FifoInputStream, ..., buf, len));
}

//
public close() {

    // Stop timer thread
    stopPeriodicTimeOutDemon(switchGETStream());
}

```

```
// Mark stream as closed  
markInputStreamClosed();
```

5

```
}
```

10

15

20

25

30

35

40

45

50

55

1b. TunnelServerSocket OutputStream – runs on the server

```

5 // Server side Protocol engine handling all requests the server receives

// Sit waiting on a ServerSocket for socket connections to come
// in ... just standard socket call
while ((socket = accept()) still open) {

10     httpheader = readHttpHeader(socket);

    if ( httpHeader.type == "GET" &&
        httpHeader.command == establish) {

15         // New socket request
        // Open an output Stream
        outputStream = socket.getOutputStream();

        // Add an entry to the database of opened Tunnel Sockets
20         addEntryTunnelSocketHashDataBase(socket, outputStream,
            httpHeader.guid);

    } else ( httpHeader.type == "GET" &&
        httpHeader.command == "switch") {

25         // Lookup tunnelSocket
        tunnelSocket =
            lookupEntryTunnelSocketDatabase(httpHeader.guid);

        // Request to switch stream generated by the TunnelSocket
        // timer on the client
        // Close the current output stream
        // Take lock to avoid conflict with
35         // tunnelOutputStreamWrite
        lock(tunnelSocket.outputStreamLock);
        close(tunnelSocket.outputStream);

        // establish a new output stream on this new connection.
40         tunnelSocket.outputStream = socket.getOutputStream();
        unlock(tunnelSocket.outputStreamLock)
    } else (httpheader.type == "POST" )
        // See pseudo-code fragment included below with Server-
        // Client direction.
45         doPOST();

    }

50     tunnelOutputStreamWrite(buf, len) {
        ...
        // Take lock to avoid conflict with "switch" in
        // tunnelServerAccept()
        lock(outputStreamLock)
55

```


EP 1 081 918 A2

```
write(outputStream, buf, len);  
unlock(outputStreamLock)
```

```
}
```

5

.....

10

15

20

25

30

35

40

45

50

55

2. Client-Server direction.5 **2a. Client side TunnelSocket OutputStream**

```

public  write(buf, length, ...) {
    // Issue an HTTP/1.0 POST request to the server
    // The on the wire format is:
    //   POST <guid>
    //   Content-Type=application/octet-stream
    //   Content-Length=<length>
    inputStream = issueHttpPostRequest(uri=guid,
        Content-Length=length,
        Content-Type="application/octet-stream",
        address=serverAddress, port=port, payload=buf);
}

```

25

2b. Server side TunnelServerSocket InputStream

```

public  readServerSideInputStream(buf, len, ...) {
    return(readFromFifo(FifoServerSideInputStream, buf, len,...));
}

...
// On receiving a POST request (see 1b above)
// copy the payload into the Fifo buffer ready for the
//   readServerSideInputStream to read it.
doPOST() {
    writeToFifo(FifoServerSideInputStream,
        httpRequest.readPayload(), httpRequest.contentLength);
}
....

```

50

Claims

55

1. A method of permitting secure access between a service external to a network firewall and a client internal to the firewall, comprising the steps of:

EP 1 081 918 A2

- (a) effecting an HTTP GET operation or equivalent thereof from the client to establish a communications socket for communicating data from the service to the client;
 - (b) after a predetermined interval effecting another GET operation or equivalent thereof to close the communications socket, irrespective of whether access between the service and the client is required to continue; and
 - 5 - repeating steps (a) and (b) while access between the service and the client is required to continue.
2. The method of claim 1, wherein the predetermined interval is less than another interval after which client software enforces termination of a communications socket established by a GET operation or equivalent thereof.
- 10 3. The method of claim 1 or claim 2, wherein information is transferred from the client to the service by an HTTP POST operation or equivalent thereof.
4. The method of any one of the preceding claims, wherein successive messages transferred between the client and the service are identified by a globally-unique identification created by the client and communicated to the service.
- 15 5. The method of claim 4, wherein the globally-unique identification is communicated via an HTTP GET or POST operation or equivalent thereof.
6. The method of claim 5, wherein the globally-unique identification is communicated in a URI relative path component.
- 20 7. The method of any one of the preceding claims, wherein communications with the client traverse a proxy service located on the client side of the firewall.

25

30

35

40

45

50

55

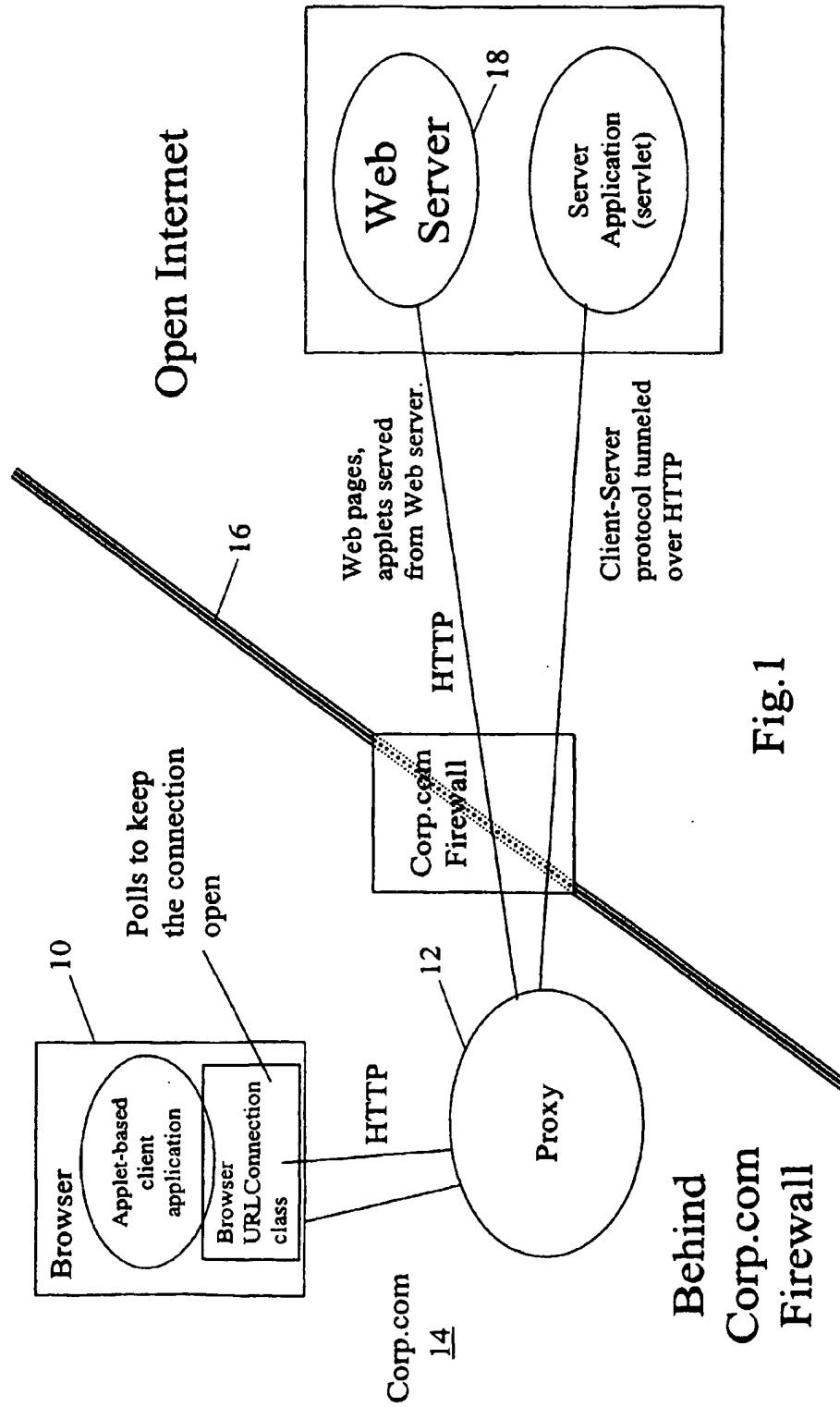


Fig.1

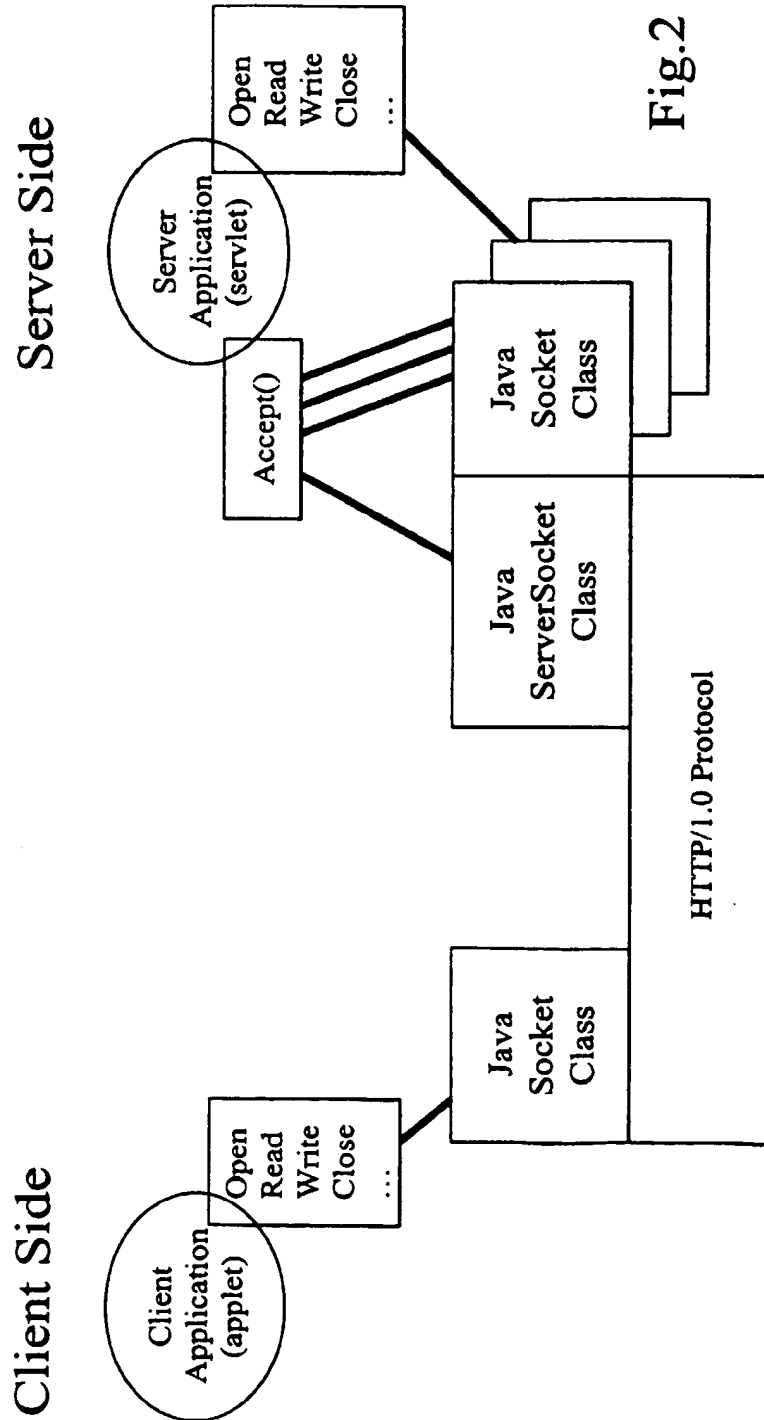


Fig.2

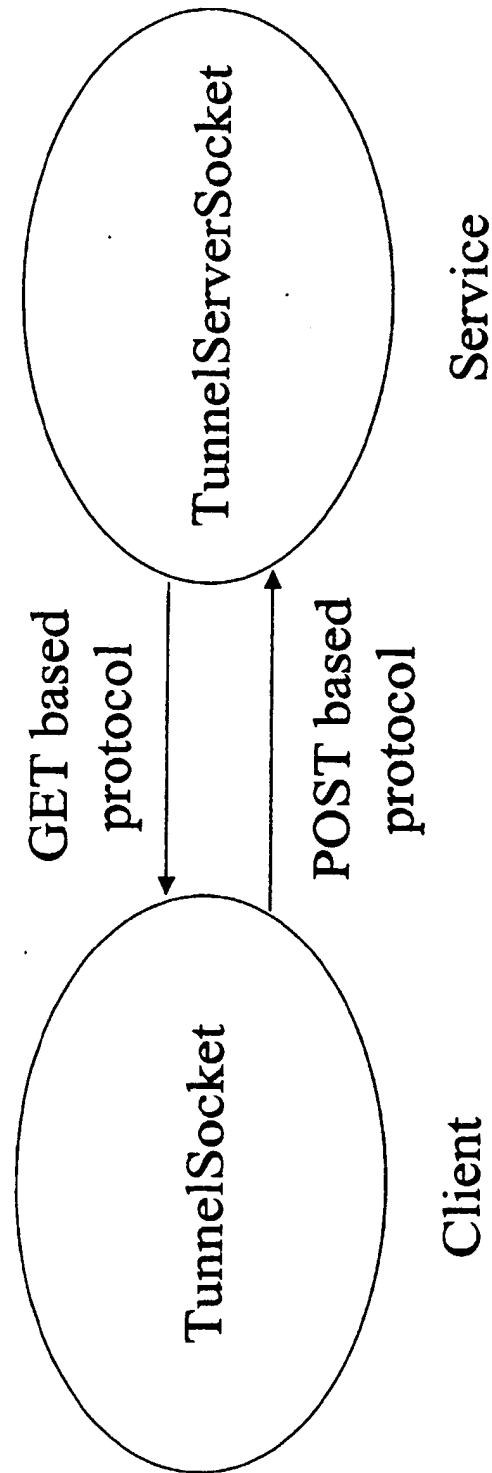


Fig.3

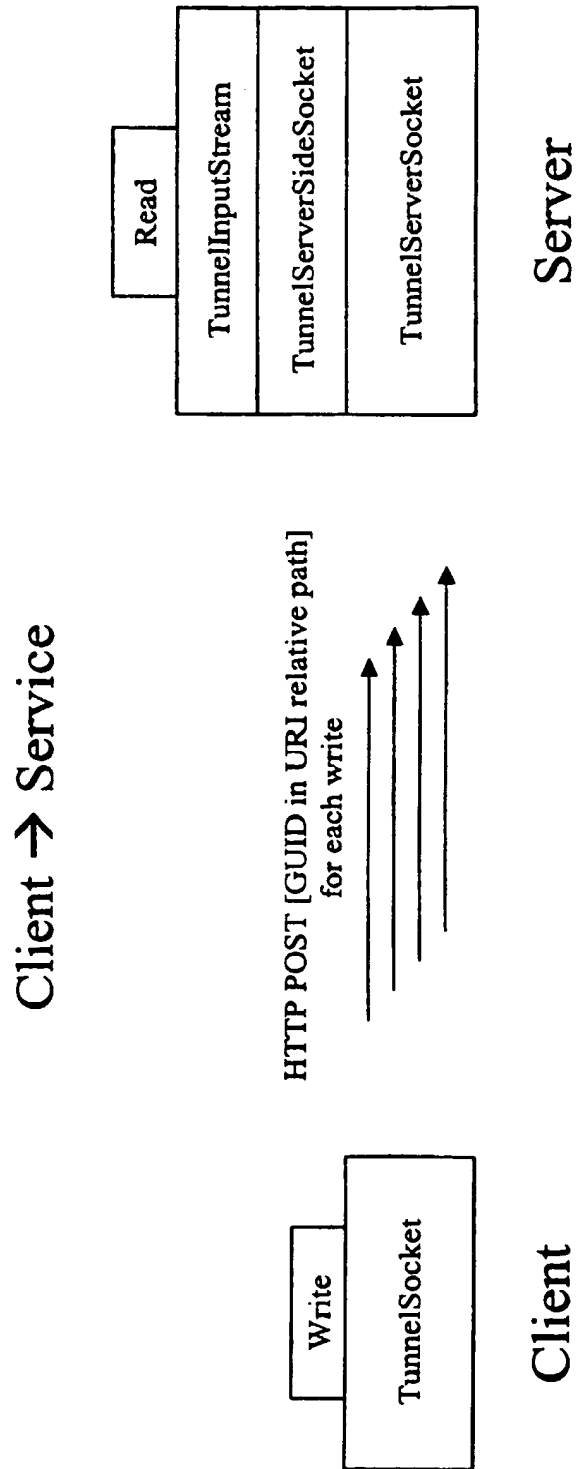


Fig.4

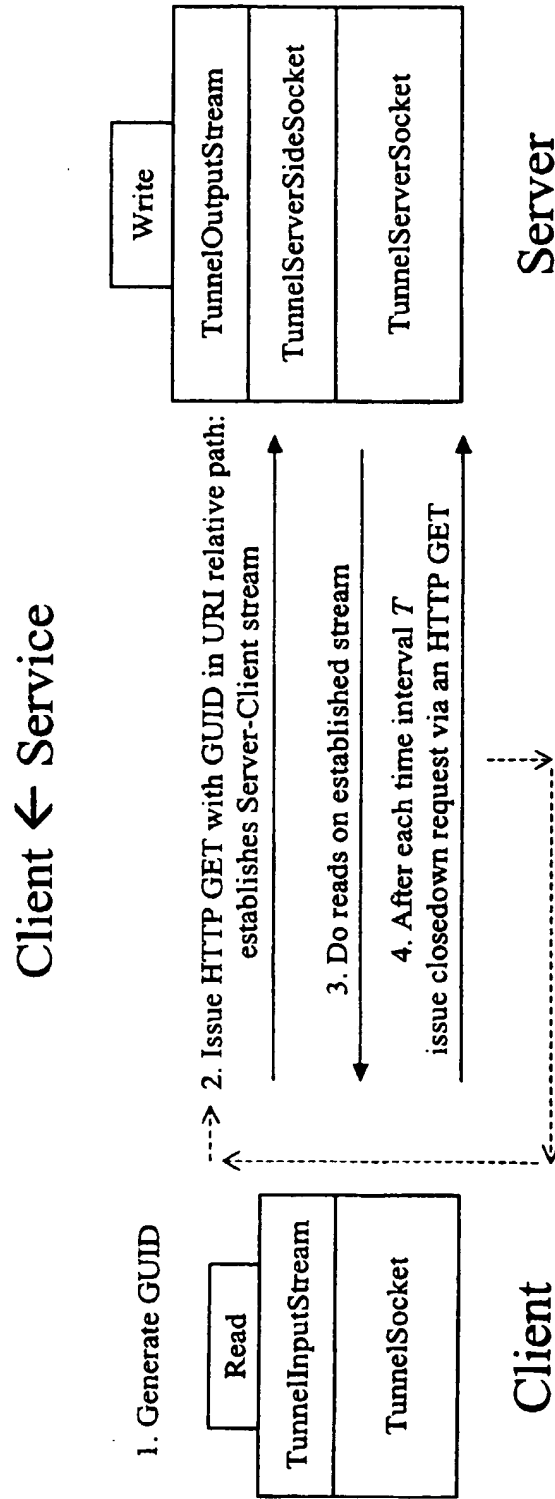


Fig.5



(12) **EUROPEAN PATENT APPLICATION**

(88) Date of publication A3:
17.12.2003 Bulletin 2003/51

(51) Int Cl.7: **H04L 29/06**

(43) Date of publication A2:
07.03.2001 Bulletin 2001/10

(21) Application number: **00307357.4**

(22) Date of filing: **25.08.2000**

(84) Designated Contracting States:
AT BE CH CY DE DK ES FI FR GB GR IE IT LI LU
MC NL PT SE
 Designated Extension States:
AL LT LV MK RO SI

- **Wilcock, Lawrence**
Malmesbury, Wiltshire SN16 9TV (GB)
- **Low, Colin**
Wotton-U-Edge, Gloucestershire GL12 7LT (GB)

(30) Priority: **04.09.1999 GB 9920834**

(71) Applicant: **Hewlett-Packard Company,**
A Delaware Corporation
Palo Alto, CA 94304 (US)

(74) Representative: **Lawrence, Richard Anthony et al**
Hewlett-Packard Limited,
IP Section,
Building 3,
Filton Road
Stoke Gifford, Bristol BS34 8QZ (GB)

(72) Inventors:
 • **Hinde, Stephen John**
Redland Bristol BS6 7DH (GB)

(54) **Providing secure access through network firewalls**

(57) To enable controlled, secure connections using a versatile protocol such as TCP/IP to be established through a firewall and proxy server, the versatile protocol is tunnelled using HTTP. Client to server communications are effected using an HTTP POST operation. Server to client communications are effected using an HTTP GET operation to establish a tunnelled socket;

this socket is closed within an interval less than any timeout imposed by the proxy server and immediately re-established by another GET operation, irrespective of whether data continue to be pending for communication to the client. A globally-unique ID is included in each POST and GET message to enable related messages to be recognized.

Client ← Service

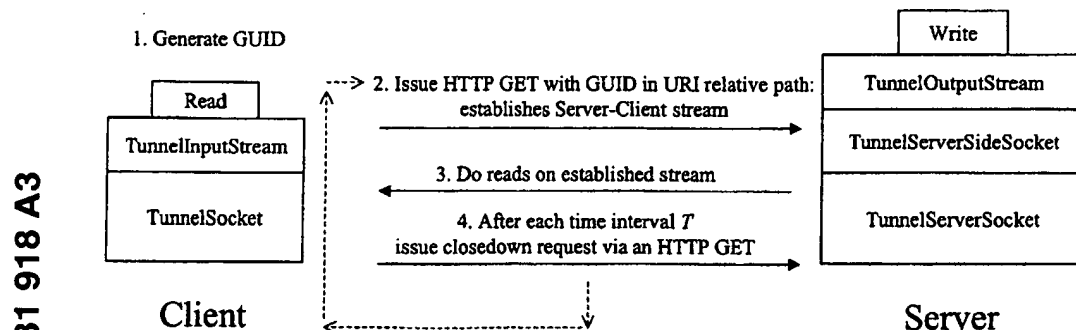


Fig.5

EP 1 081 918 A3



European Patent
Office

EUROPEAN SEARCH REPORT

Application Number
EP 00 30 7357

DOCUMENTS CONSIDERED TO BE RELEVANT			
Category	Citation of document with indication, where appropriate, of relevant passages	Relevant to claim	CLASSIFICATION OF THE APPLICATION (Int.Cl.7)
P,X	US 6 169 992 B1 (APPELBAUM MATTHEW A ET AL) 2 January 2001 (2001-01-02) * abstract * * column 3, line 21 - column 5, line 62 * * column 18, line 30 - column 37, line 35 *	1-7	H04L29/06
A	--- HUNT R: "Internet/Intranet firewall security-policy, architecture and transaction services" COMPUTER COMMUNICATIONS, BUTTERWORTHS & CO. PUBLISHERS LTD, GB, vol. 21, no. 13, 1 September 1998 (1998-09-01), pages 1107-1123, XP004146571 ISSN: 0140-3664 * abstract * * page 1107, right-hand column, line 18 - page 1108, left-hand column, line 47 * * page 1116, left-hand column, line 30 - page 1117, left-hand column, line 5 * * page 1117, right-hand column, line 31 - page 1119, left-hand column, line 4 * -----	1-7	<div>TECHNICAL FIELDS SEARCHED (Int.Cl.7)</div> <div>H04L</div>
The present search report has been drawn up for all claims			
Place of search THE HAGUE		Date of completion of the search 29 October 2003	Examiner Adkhis, F
<div>CATEGORY OF CITED DOCUMENTS</div> <div> X : particularly relevant if taken alone Y : particularly relevant if combined with another document of the same category A : technological background O : non-written disclosure P : intermediate document T : theory or principle underlying the invention E : earlier patent document, but published on, or after the filing date D : document cited in the application L : document cited for other reasons & : member of the same patent family, corresponding document </div>			

EPO FORM 1503 (01.02.02) (P04/C01)

(12) **EUROPEAN PATENT APPLICATION**

(43) Date of publication:
05.04.2000 Bulletin 2000/14

(51) Int Cl.7: **H04L 29/08, H04L 29/06,
H04L 12/56**

(21) Application number: **99307651.2**

(22) Date of filing: **28.09.1999**

(84) Designated Contracting States:
**AT BE CH CY DE DK ES FI FR GB GR IE IT LI LU
MC NL PT SE**
Designated Extension States:
AL LT LV MK RO SI

(72) Inventors:
• **Chapman, Alan Stanley John**
Kanata, Ontario K2K 1V5 (CA)
• **Kung, Hsiang-Tsung**
Lexington, MA 02173 (US)

(30) Priority: **30.09.1998 CA 2249152**

(74) Representative: **Funnell, Samantha Jane et al**
Hepworth Lawrence Bryer & Bizley
Merlin House
Falconry Court
Bakers Lane
Epping, Essex CM16 5DQ (GB)

(71) Applicant: **Nortel Networks Corporation**
Montreal, Quebec H2Y 3Y4 (CA)

(54) **Apparatus for and method of managing bandwidth for a packet based connection**

(57) Flow control of packet based traffic by window is known. Novel modification is described which causes the flow control mechanism to reduce sending rate to some configured number rather than just reducing it by a fixed amount such as one half. The description also shows how the flow control mechanism can be constrained to a maximum rate. The configured numbers will assure that the connection can always run at a minimum rate but not more than a maximum rate. If the

guaranteed minimum bandwidth is known and the round trip time between the end points is known or has been calculated, then the sender node needs only reduce its window to that which corresponds to a sending rate equal to that configured number. In this way the protocol will still probe for extra, opportunistic bandwidth but will be able to maintain the minimum rate. In a similar way a window that corresponds to the maximum rate can be calculated and used to constrain the maximum rate of sending.

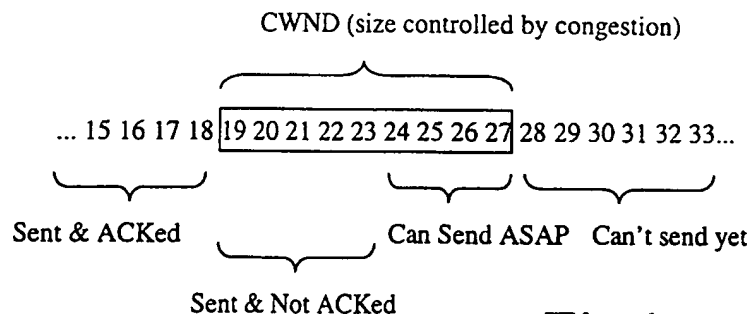


Fig 1

EP 0 991 244 A2

Description**Field of Invention**

[0001] The invention resides generally in the field of digital data transmission through a network. In particular, it relates to an apparatus for and method of transmitting digital data in streams of packets while observing guaranteed minimum and/or maximum bandwidth allocations.

Background of Invention

[0002] In contrast to circuit based transport systems, a packet based transport system allows the access bandwidth to be dynamically allocated. Remote nodes can be represented as logical ports but there is no commitment of bandwidth when this is not needed. The physical access link is fully available for traffic to any destination. In a packet transport system, virtual pipes are provided between any two transport access points. These pipes may be guaranteed some minimum rate of transmission but certainly it is required that an access point can make opportunistic use of spare bandwidth up to some maximum amount. Frame relay as a packet based transport allows more efficient use of bandwidth by permitting statistical multiplexing of data streams, thus allowing it to exploit unused bandwidth. However, there is no mechanism (protocol) to ensure reliable delivery of the frames and under congestion conditions frames are discarded and the higher layer protocols must compensate for the loss. The discard of frames is not sensitive to the impact on the higher layer protocols and the frame flows do not directly adapt to the network conditions. ATM with an effective flow control can provide a lossless but dynamic transport. However, it relies on some reasonable level of complexity at the transport switching points to achieve the flow control, the effectiveness of which has not yet been proven in the field. ATM without flow control requires that cells be discarded under congestion conditions and this discard should be aware of packet boundaries and the impact on the higher layer protocol such as TCP protocol. These issues are just beginning to be understood.

[0003] It is commonly understood in the field of the present invention that a layer under the networking layer is called "transport" layer and provides pipes between networking layer nodes. This is in contrast to the layered model of the OSI (open systems interconnect) in which the transport layer resides upon the network layer, which in turn sits on top of the data link layer. The data link layer provides similar functionalities to those of the transport layer of the present description. Throughout the specification, the former designation is used.

[0004] Therefore in the TCP/IP model, IP layer resides under TCP layer. The IP layer is the network layer in which IP (Internet protocol) runs. An internetwork differs from a single network, because different parts may

have wildly different topologies, bandwidth, delays, packet sizes, and other parameters. The TCP layer is the transfer layer in which the TCP (transmission control protocol) runs. The TCP has been used for ensuring reliable transfer of byte stream data between two end points over an internetwork, which may be less reliable. TCP allows a shared and adaptive use of available bandwidth of a transmission link between two end points. It does this by having the sender gradually increase the rate of sending until a packet is lost whereupon it reduces its rate significantly and repeats the gradual increase. Thus, TCP tends to give connections their proportional share of the available bandwidth on a link although different connection characteristics can cause large variations in the sharing.

[0005] In Internet terminology, aggregating traffic streams by encapsulating them into a single IP stream is often called tunneling. Applicant's copending patent application Serial No. 09/066,888 filed on Apr. 28, 1998 describes an invention, which re-uses TCP in a packet based transport to provide TCP tunneling which can conveniently be called "TCP trunking". The use of TCP provides for reliable delivery of data between two transport access points while permitting that transport to offer elasticity and bandwidth sharing. Aggregating traffic streams into TCP tunnels reduces the size of buffers and tables in the transport switches. TCP is well suited to the use of first-in-first-out queues and allows simple implementations at the switching nodes. TCP is also inherently provides for resequencing of out-of-order packets which can occur when switching nodes spread load over multiple links.

[0006] It is expected that in future networks, particularly those using TCP trunking between aggregation points, it will be required to assure a minimum sending rate for the connection while still allowing the connection to probe for more throughput if it is available. This opportunistic acquisition of bandwidth might be limited to some maximum. TCP currently has no capability to support a guaranteed minimum bandwidth or a maximum permitted bandwidth but this will be essential as IP networks introduce virtual private networks with performance guarantees. The present invention offers a good solution to such a problem.

[0007] It should be noted that in a carrier owned transport network the TCP functionality would be in transport access points rather than host computers, the variability of paths will be low and parameters such as connection round trip time will be very stable. These features make it much easier to envisage modifications to the TCP protocol for these networks. However, the use of this invention can be more generally applied to other TCP hosts and other sliding window protocols.

[0008] Many flows on today's networks carry only a small amount of information, say ten or twenty packets, and the rate adaptation feature of TCP is not as relevant as the reliability feature. Providing reliability without incurring TCP time-outs on the loss of a single packet

would greatly improves user throughput and network efficiency.

Objects of Invention

[0009] It is therefore an object of the invention to provide sliding window based flow control for point-to-point connections in a packet network, which constrain the minimum and/or maximum bandwidths available to that connection.

[0010] It is another object of the invention to provide modifications to the TCP protocol to provide for constraining the minimum and/or maximum bandwidths achieved by the connection.

[0011] It is yet an object of the invention to ensure the absence of TCP retransmission timeouts for flow with small windows by maintaining a minimum rate of sending.

Summary of Invention

[0012] Briefly stated, in accordance with one aspect, the invention is directed to a method of sending data in packets via a connection by way of sliding window algorithm in which a flow of data into the connection is controlled in response to acknowledged packets and the connection observing either or both of a guaranteed minimum bandwidth and a maximum permitted bandwidth. The method comprises steps of (1) calculating a congestion window hereinafter called C-WND of the connection, (2) calculating either or both of a guaranteed minimum bandwidth window hereinafter called MIN-WND and a maximum permitted bandwidth window hereinafter called MAX-WND. The method further includes steps of (3) determining if the MIN-WND or MAX-WND is invoked on the connection, based on their relationship with C-WND and (4) allowing the transmission of one or more packets of data into the connection if either MIN-WND or MAX-WND permits said transmission.

[0013] According to another aspect, the invention is directed to an apparatus for sending data in packets via a connection by way of sliding window algorithm in which a flow of data into the connection is controlled in response to acknowledged packets and the connection observing either or both of a guaranteed minimum bandwidth and a maximum permitted bandwidth. The apparatus comprises a flow control module for controlling a flow of packets into the connection in response to acknowledged packets, a congestion window arithmetic module for calculating a congestion window hereinafter called C-WND of the connection and a bandwidth monitoring window arithmetic module for calculating either or both of a guaranteed minimum bandwidth window hereinafter called MIN-WND and a maximum permitted bandwidth window hereinafter called MAX-WND. The apparatus further includes control logic module for determining if the MIN-WND or MAX-WND is invoked on the connection, a transmitter for transmitting a series of

packets of data into the connection and a controller for allowing the transmission of one or more packets of data into the connection if either MIN-WND or MAX-WND permits said transmission.

Brief Description of Drawings

[0014]

Figure 1 is an illustration showing the nature of the sliding window algorithm.

Figure 2 shows mechanisms of acknowledgements.

Figure 3 shows that the window is inflated and moved when a non-duplicate ACK is received.

Figure 4 shows the window when a packet is lost.

Figure 5 shows the nature of MIN-WND according to an embodiment of the invention.

Figure 6 shows the MIN-WND when a duplicate ACK is received.

Figure 7 shows the MIN-WND when a non-duplicate ACK is received.

Figure 8 shows the nature of MAX-WND according to an embodiment of the invention.

Figure 9 shows the MAX-WND when a duplicate ACK is received.

Figure 10 shows the MAX-WND when a non-duplicate ACK is received.

Figures 11-14 show the relationships of various windows.

Figure 15 is a block diagram of a TCP node.

Detailed Description of Preferred Embodiments of Invention

[0015] When providing services with bandwidth guarantees, a packet transport network should be able to emulate the circuit-based mesh in that a defined minimum bandwidth can be allocated between any pair of nodes. However, unused bandwidth should be made available to other flows in a dynamically shared fashion so that a flow can opportunistically exceed its minimum. In some cases it is also useful to implement a maximum limit on how much extra bandwidth a pair of nodes can use.

[0016] The conventional IP network implements bandwidth sharing among host machines using the transport control protocol (TCP). In TCP the sender (sender host machine) constantly tests the network to see if more bandwidth is available and uses the loss of a packet determined by sequence numbers of TCP packets as an indication to decrease its rate. Any lost packets are sent again so that there is a reliable flow of traffic. The loss of too many packets can cause the TCP connection to enter the timed out state. Consecutive timeouts are increased in an exponential way until eventually the connection is closed.

[0017] The general characteristic of TCP is that it is

self-clocking. That is to say, the sender will wait for an acknowledgment from the receiver for the packets already sent before sending more packets. If the sender waited for each individual packet to be acknowledged then the maximum rate that the connection could achieve would be one packet per round trip time of the connection. To increase the sending rate while keeping the self clocking nature of the protocol, the sender is allowed to send some number of packets while waiting for an earlier packet to be acknowledged. This number of packets is called the window. The receiver itself may constrain the size of the window in order to limit its buffer requirement.

[0018] Each packet contains a sequence number, which increases according to the number of bytes transmitted. The receiver acknowledges packets using this numbering scheme and always acknowledges the latest packet received in correct sequence. It may acknowledge each packet individually or wait in order to reduce overhead (this is called Delayed ACK). It should definitely send an acknowledgment at least every second packet. If a packet is received which is not in correct sequence the receiver will immediately send an acknowledgment but the sequence number it acknowledges will be that of the last packet which was received in the correct sequence. It should be noted that the sequence number in a packet corresponds to the last byte in the packet and the acknowledgment contains the next expected in-sequence byte number and thus acknowledges all bytes up to that number. In general terminology a packet is acknowledged when the receiver reports that the next expected byte number is later than any bytes contained in that packet.

[0019] The maximum rate of sending on a TCP connection is equal to the window size divided by the round trip time of the connection. TCP will constantly try to increase its rate by increasing the window size. When a packet is lost the window size is reduced and the gradual increase is begun again. The current size of the window is called the congestion window (C-WND) and can vary between one packet and the maximum that the receiver is prepared to accept (R-WND: receiver window).

[0020] Figure 1 shows the nature of the sliding window. The window reflects the data sent but not yet acknowledged as well as the amount of data that can still be sent without waiting for an acknowledgement. As a packet is acknowledged the window advances so that the left-hand side is equal to the earliest unacknowledged byte number. The right hand side of the window is equal to the highest byte sequence number that can be sent before the transmitter must wait for further acknowledgements. It should be noted that the receiver will only acknowledge bytes received which are in a complete sequence. Later bytes that have been received will not be acknowledged until all previous bytes have been received.

[0021] Packet loss is detected in one of two ways. If the sender does not get an acknowledgment within a

certain time (TCP retransmission time-out) it will assume that a packet has been lost and will reduce its C-WND size to one packet as well as resending the lost packet. If the sender sees multiple acknowledgments (called duplicate ACK) of the same packet it can decide that packet has been lost even before the retransmission time-out occurs. Many TCP implementations include this fast retransmission and recovery capability. The window size is cut in half and the lost packet is retransmitted. Avoiding time-out gives a great boost to perceived performance but it is only effective when the window is large enough to allow enough duplicate acknowledgments to be generated (usually three). This is shown schematically in Figure 2 in which packet 19 is lost and the receiver acknowledges reception of packets up to 18. Multiples of acknowledged packets indicate that the receiver still expects packet 19. For windows smaller than about five packets, it is not possible to guarantee that fast retransmission will be invoked. Many flows on today's networks carry only a small amount of information, say ten or twenty packets and never develop large windows. Thus, the loss of a single packet can force the connection into timeout.

[0022] There are two operations on the C-WND and they are shown in Figures 3 and 4 respectively:

(a) Inflate Window (Figure 3)

When a non-duplicate ACK is received, C-WND is inflated by extending the window's right edge and moves to the right so that the first byte in the window is the earliest unacknowledged byte. The inflation factor is a function of the TCP implementation.

(b) Deflate Window (Figure 4)

When packet loss occurs the window is reduced in size by retracting the window's right edge and the packet is retransmitted.

[0023] The transport system is required to provide some minimum level of bandwidth for the total traffic between any pair of access points. As mentioned earlier, usually TCP will reduce its sending rate very aggressively when a packet is lost. It is envisaged however that TCP can be modified to cause it to reduce sending rate to some configured number rather than just reducing it by a fixed amount such as one half. The configured number will assure that the connection can always run at a minimum rate. It is also envisaged in some instances that TCP can be constrained to a maximum rate, less than it would achieve normally so that a connection would not occupy all the available bandwidth.

[0024] Therefore, if the guaranteed minimum bandwidth is known and the round trip time (RTT) between the end points is known or has been estimated, then the TCP sender node needs only reduce its window to that which corresponds to a sending rate equal to that configured number. In this way the protocol will still probe for extra, opportunistic bandwidth but will be able to

maintain the minimum rate. Similarly sending will be inhibited when the TCP window reaches a size corresponding to the maximum bandwidth.

[0025] This invention introduces the concept of overlay windows. This concept makes it easy to understand the design intent and permits the modification to be added without having to make substantial changes to the main body of standard TCP operating code.

[0026] According to one embodiment, TCP is modified to cause it to constrain its sending rate to be between some configured minimum and/or maximum numbers rather than between one packet and the receiver window size (R-WND). This modification is only needed at the TCP transmitter. The configured minimum number will assure that the connection can always run at a minimum rate and the configured maximum number prevents all the available bandwidth of the connection from being taken by a node pair. The modification to the TCP transmitter will also improve TCP's resilience in the sense that the connection will not experience exponentially increasing time out under packet loss. This improved resilience against packet loss is achieved without loading the network more than the desired guaranteed minimum bandwidth for the TCP connection, or one packet per the round trip time of the connection. As well as providing for a guaranteed minimum rate during the lifetime of a connection, this modification can also be enabled selectively to prevent time-out at the times when the window size of the connection is too small to allow fast retransmission and recovery.

[0027] As mentioned earlier, in normal TCP the sender is allowed to send some number of packets while waiting for an acknowledgment of an earlier packet and this number is referred to as the window. Arithmetically, one can see that the maximum rate that a connection can achieve is equal to the window size divided by the round trip time of the connection (RTT) in seconds. To assure a guaranteed minimum bandwidth (GMB bytes per second) it is necessary that the connection can always send at least GMB times RTT bytes in any RTT period and that the connection is not stalled by lost packets but will keep sending unless the normal keepalive process closes the connection.

[0028] According to one embodiment, the TCP transmitter uses following variables:

GMB: The guaranteed minimum bandwidth in bytes per second. This is a new, configured parameter.

MPB: The maximum permitted bandwidth in bytes per second. This is a new, configured parameter.

RTT: The estimated round trip time of the connection in seconds.

R-WND: The maximum window size acceptable to the receiver (as advertised by the receiver in conjunction with acknowledgements).

C-WND: The size of the congestion window as computed by the existing TCP algorithm.

MIN-WND: This is a sliding window based on the

guaranteed minimum bandwidth. This is a new, computed variable

MAX-WND: A sliding window based on the permitted maximum bandwidth. This is a new, computed variable.

Pkt: Packet size is the packet payload size currently used by the connection, in bytes.

RTO: TCP retransmission time-out value is the period, in seconds, after which, in the absence of acknowledgments, unacknowledged packets will be retransmitted. (Typically RTO is 1 sec or greater.)

OwT: Out-of-window send timer, in seconds, defines the time after which a packet can be sent even if the state of the TCP congestion window would normally not allow it. (A suggested value of OwT is 0.2 secs.). This is a new configured parameter.

RsT: Resend timer period, in seconds, has a value larger of OwT, RTT,

Pkt/GMB but always smaller than RTO. This is a new, computed variable.

[0029] According to an embodiment of the invention, the guaranteed minimum bandwidth for a TCP connection is achieved as follows.

[0030] While the connection is open the transmitter can send one packet into the network if it is allowed by the sliding window advertised by the receiver and if any one of the following conditions are met:

C1: The transmitted packet is allowed by the normal TCP congestion window and not disallowed by MAX-WND

C2: The transmitted packet is allowed by MIN-WND

C3: RsT expires.

[0031] Figure 5 shows the nature of MIN-WND, which is very similar to C-WIND except that the size does not change according to congestion but is tied to the value of GMB and RTT.

[0032] There are two operations on the MIN-WND and they are shown in Figures 6 and 7:

(a) Inflate Window (Figure 6)

When a duplicate ACK is received, MIN-WND is inflated by 1 Pkt by extending the window's right edge.

(b) Reset Window Size and Move Right (Figure 7)
When a non duplicate ACK is received, MIN-WND resets to its original size (GMB*RTT), and moves to the right so that the first byte in the window is the earliest unacknowledged byte.

[0033] As seen in the figures, when a duplicate acknowledgement is received the window is inflated by one packet. The reception of an acknowledgment shows that the receiver has received a packet even if it was not in sequence. The inflation of the window ensures that the connection can continue sending new packets even when an acknowledgement is missing. A

new packet will be sent for each duplicate acknowledgement received. This prevents the connection stalling but does not increase the number of packets in the network. However, as soon as a non-duplicate acknowledgement is received the window is reset to the normal size.

[0034] Similarly Figure 8 shows the format of MAX-WND which is identical to MIN-WND except that the size is based on the maximum permitted bandwidth.

[0035] Like the windows described thus far, there are two operations on the MAX-WND and they are shown in Figures 9 and 10:

(a) Inflate Window (Figure 9)

When a duplicate ACK is received, MAX-WND is inflated by 1 Pkt by extending the window's right edge.

(b) Reset Window Size and Move Right (Figure 10)

When a non duplicate ACK is received, MAX-WND resets to its original size ($MPB \cdot RTT$), and moves to the right so that the first byte in the window is the earliest unacknowledged byte.

[0036] Figures 9 and 10 therefore show that the window is also inflated when duplicate acknowledgements are received and reset when a non-duplicate acknowledgement is seen.

[0037] Figures 11 to 14 show unmodified TCP algorithm and the overlay methods of modifying the TCP algorithm, according to embodiments of the invention. In these embodiments, the chosen window is defined as being the maximum acceptable window after taking into account the requirements of the overlay rules. Therefore, Figure 11 shows unmodified TCP where the chosen window is the normal congestion window and can have a value between 1 and R-WND. Figure 12 on the other hand shows how, when the highest sequence number within C-WND falls below that of MIN-WND, the GMB overlay takes effect and the chosen window is equal to MIN-WND. Similarly in Figure 13, when the highest sequence number of C-WND becomes greater than that of MAX-WND, the MPB overlay takes effect and chosen window becomes equal to MAX-WND. Figure 14 shows how MIN-WND and MAX-WND are overlaid on the normal TCP algorithm to provide the complete bandwidth control.

[0038] When the rate of sending is between the configured limits the normal TCP algorithm controls the rate. If the rate tends to fall below the minimum then MIN-WND comes into play. If the rate reaches the maximum then MAX-WND takes effect. The normal TCP mechanism is still in control of reliability and of elasticity within the configured limits. An overriding timer RsT ensures that even when no acknowledgements are being received, a minimum rate of packets are still sent to stimulate acknowledgements and eventually retransmission without being stalled by TCP timeout.

[0039] Figure 15 illustrates schematically in block diagram a TCP node according to one embodiment of the

invention. The node is connected to a network and includes a transmitter and receiver of an IP module 20. The customer data 22 is processed by a TCP module 24 which forms the data into TCP packets before the transmitter send them into the network. In receive direction, of course, the module extracts the customer data and transfers it to the customer's terminal for outputting. The clock 26 generates clock signals which times a variety of operations of the node. The arithmetic module 28 is shown in a separate box which performs computations described thus far under control of the control logic 30. Controller 32 supervises the over-all operation of TCP module.

15 Assertions:

[0040]

A1. The TCP connection is allowed to transmit at least $GMB \cdot RTT$ bytes per RTT except while recovering from missing acknowledgements.

A2. The TCP connection will not stop transmitting for a period longer than RsT under any circumstances of packet loss, until a normal TCP timeout causes the connection to close.

A3. The loading of the network by the transmitter is at most the maximum of GMB and Pkt/RsT , unless more is allowed by the TCP congestion window within the limits of MPB. (RsT is always greater than or equal to RTT)

A4. In the presence of sufficient bandwidth, the TCP connection is able to sustain a maximum rate of $MPB \cdot RTT$ bytes per RTT except while recovering from missing acknowledgements.

[0041] It should be noted that it is a necessary requirement that an admission process will limit the number of TCP connections sharing a network link, so that the sum of GMBs for all the TCP connections including TCP and IP header overheads is no more than the link bandwidth.

[0042] It should also be noted that even when a guaranteed minimum bandwidth is not wanted for the whole duration of a flow, the application of this modification would improve performance for short flows of less than say ten or twenty packets by preventing TCP time-out which normally occurs after a single packet loss. In this case the modification could be enabled while the flow was in a fragile state and turned off once a certain number of packets were successfully sent.

Claims

1. A method of sending data in packets via a connection by way of sliding window algorithm in which a flow of data into the connection is controlled in response to acknowledged packets and the connection observing either or both of a guaranteed mini-

mum bandwidth and a maximum permitted bandwidth, comprising steps of:

- (1) calculating a congestion window hereinafter called C-WND of the connection; 5
 - (2) calculating either or both of a guaranteed minimum bandwidth window hereinafter called MIN-WND and a maximum permitted bandwidth hereinafter called MAX-WND; 10
 - (3) determining if the MIN-WND or MAX-WND is invoked on the connection, based on their relationship with C-WND; and
 - (4) allowing the transmission of one or more packets of data into the connection if either MIN-WND or MAX-WND permits said transmission. 15
2. The method of sending data in packets via a connection by way of sliding window algorithm, according to claim 1 wherein step (3) is performed by comparing either C-WND and MIN-WND or C-WND and MAX-WND. 20
 3. The method of sending data in packets via a connection by way of sliding window algorithm, according to claim 2 comprising a further step of: 25
 - inflating either or both MIN-WND and MAX-WND in response to each duplicate acknowledgment.
 4. The method of sending data in packets via a connection by way of sliding window algorithm, according to claim 3 comprising further steps of: 30
 - (5) counting a reset timer hereinafter called RsT in connection with the congestion window; and 35
 - (6) allowing the transmission of one or more packets of data into the connection if at least one of C-WND, MIN-WND and RsT permits said transmission. 40
 5. The method of sending data in packets via a connection by way of sliding window algorithm, according to claim 3 wherein steps (3) and (4) is performed only during a portion of the connection period. 45
 6. The method of sending data in packets via a connection by way of sliding window algorithm, according to claim 4 wherein steps (3) and (4) is performed only during a portion of the connection period. 50
 7. The method of sending data in packets via a connection by way of sliding window algorithm, according to claim 3 comprising further steps of: 55
 - (7) counting a reset timer hereinafter called RsT in connection with the congestion window;

and

(8) allowing the transmission of one or more packets of data into the connection if at least one of C-WND, MAX-WND and RsT permits said transmission.

8. The method of sending data in packets via a connection by way of sliding window algorithm, according to claim 7 wherein steps (3) and (4) is performed only during a portion of the connection period.
9. The method of sending data in packets via a connection by way of sliding window algorithm, according to claim 4 wherein the connection is a TCP connection.
10. The method of sending data in packets via a connection by way of sliding window algorithm, according to claim 7 wherein the connection is a TCP connection.
11. An apparatus for sending data in packets via a connection by way of sliding window algorithm in which a flow of data into the connection is controlled in response to acknowledged packets and the connection observing either or both of a guaranteed minimum bandwidth and a maximum permitted bandwidth, comprising
 - a flow control module for controlling a flow of packets into the connection in response to acknowledged packets;
 - a congestion window arithmetic module for calculating a congestion window hereinafter called C-WND of the connection;
 - a bandwidth monitoring window arithmetic module for calculating either or both of a guaranteed minimum bandwidth window hereinafter called MIN-WND and a maximum permitted bandwidth hereinafter called MAX-WND;
 - control logic module for determining if the MIN-WND or MAX-WND is invoked on the connection;
 - a transmitter for transmitting a series of packets of data into the connection; and
 - a controller for allowing the transmission of one or more packets of data into the connection if either MIN-WND or MAX-WND permits said transmission.
12. The apparatus for sending a packet of data via a connection through a network, according to claim 11, further comprising:
 - a packetizing module for packetizing data to be sent into a series of packets, each having an individual sequence number; and
 - a clock for timing operations of various mod-

ules.

13. The apparatus for sending a packet of data via a connection through a network, according to claim 12, wherein the packet of data is a TCP packet of data and the apparatus further comprising: 5

a TCP protocol module for forming series of TCP packets of data, each having a sequence byte number; and 10

the flow control module for controlling a flow of packets into the connection in response to acknowledged sequence byte numbers of TCP packet 15

14. The apparatus for sending a packet of data via a connection through a network, according to claim 13, wherein the flow control module comprises further a window control module for adjusting the size of either or both MIN-WND and MAX-WND in response to each duplicate ACK. 20

25

30

35

40

45

50

55

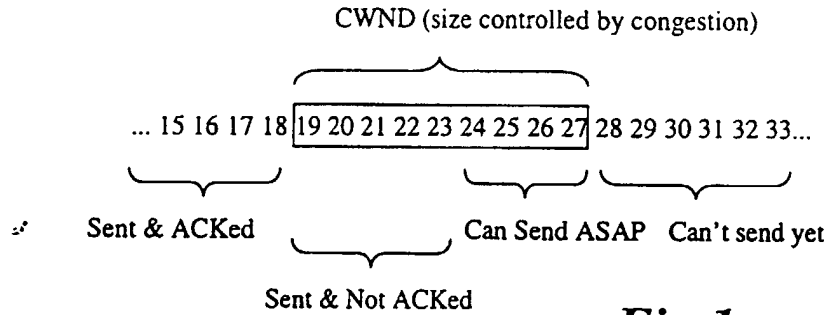


Fig 1

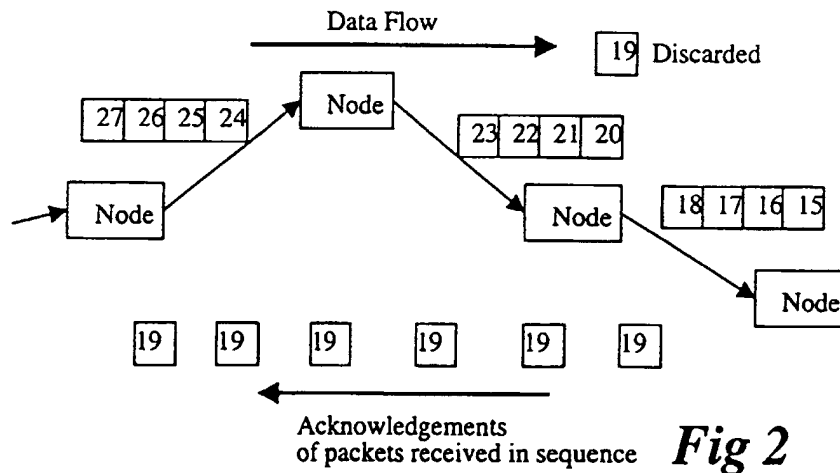


Fig 2

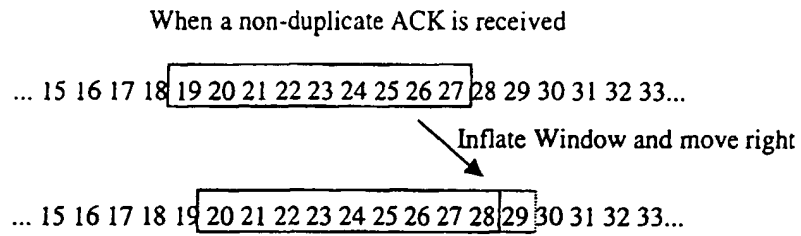


Fig 3

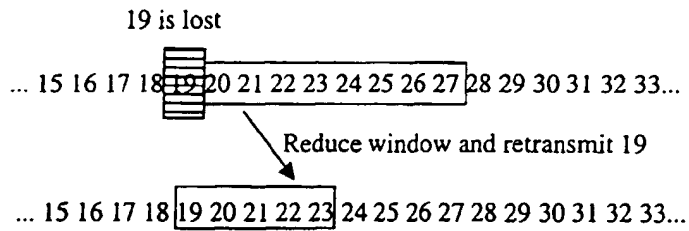


Fig 4

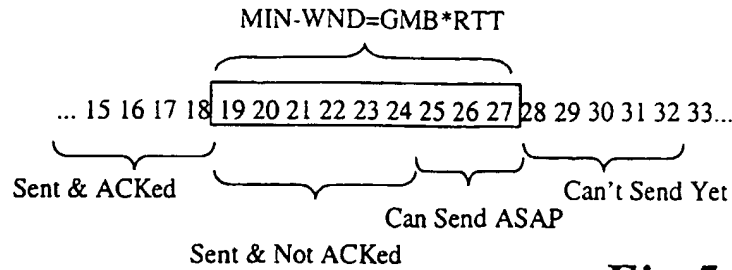


Fig 5

When a duplicate ACK is received

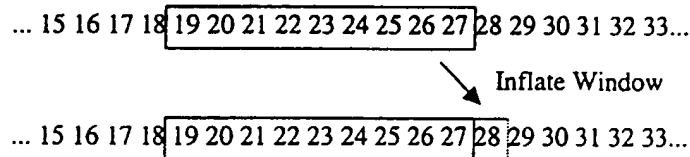


Fig 6

When a non-duplicate ACK is received

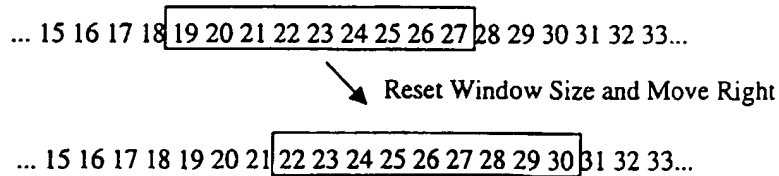


Fig 7

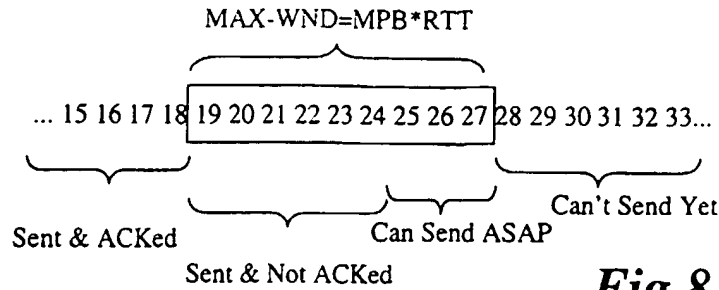


Fig 8

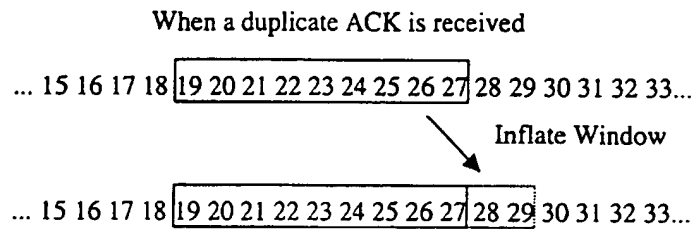


Fig 9

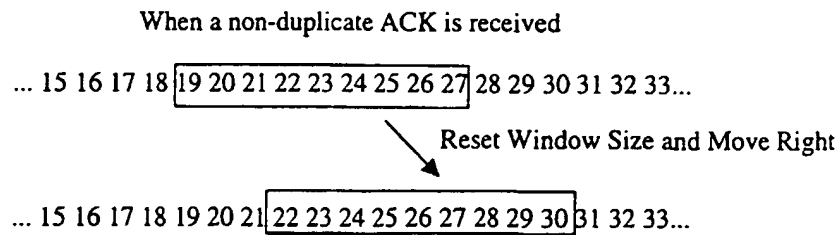


Fig 10

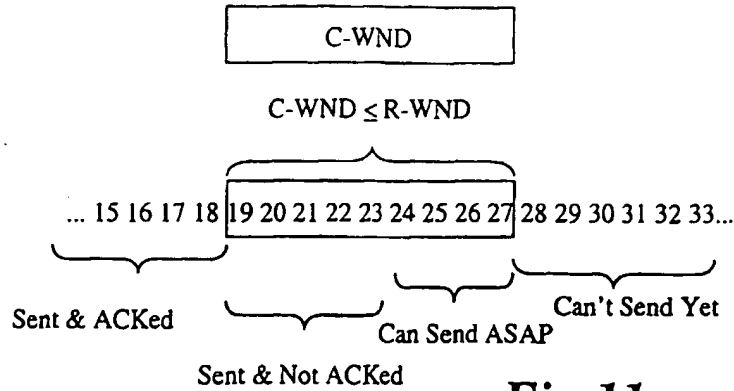


Fig 11

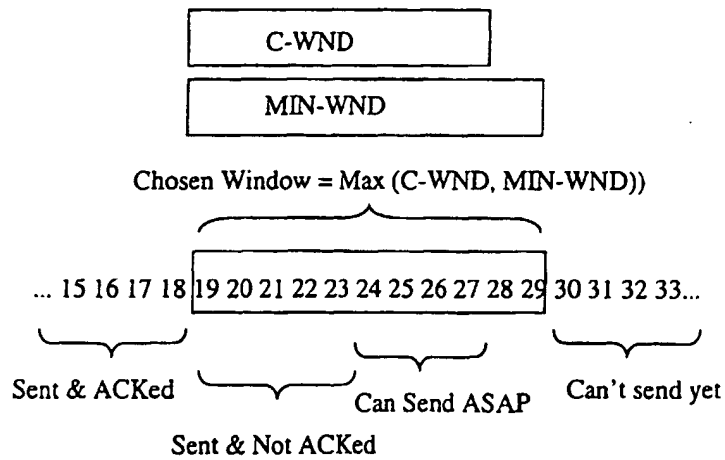


Fig 12

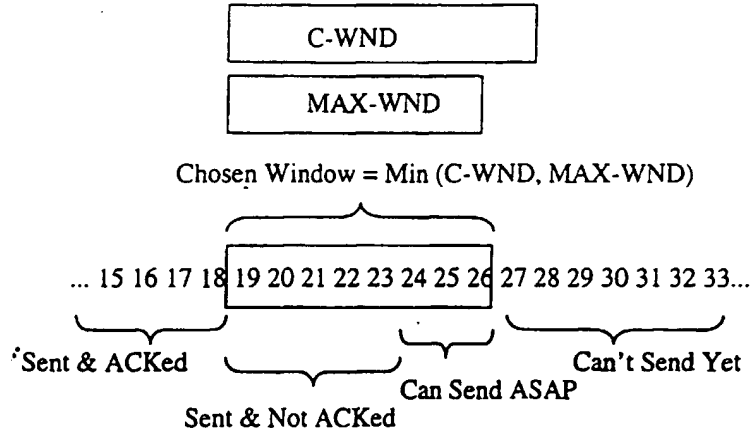


Fig 13

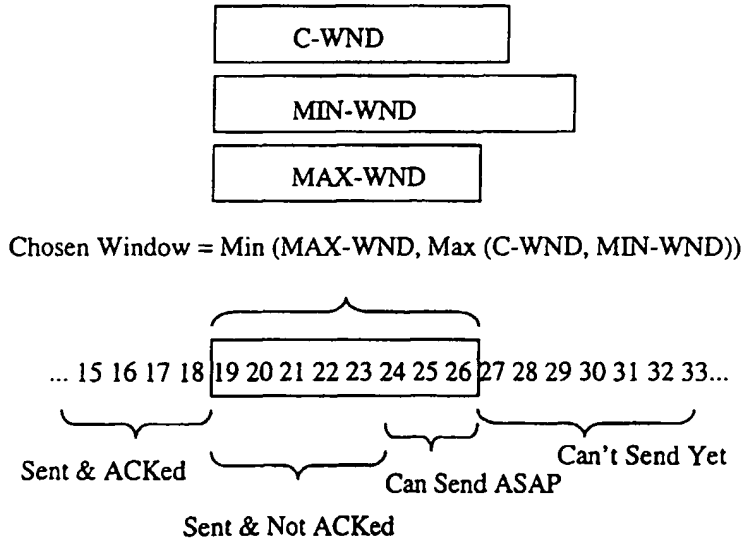


Fig 14

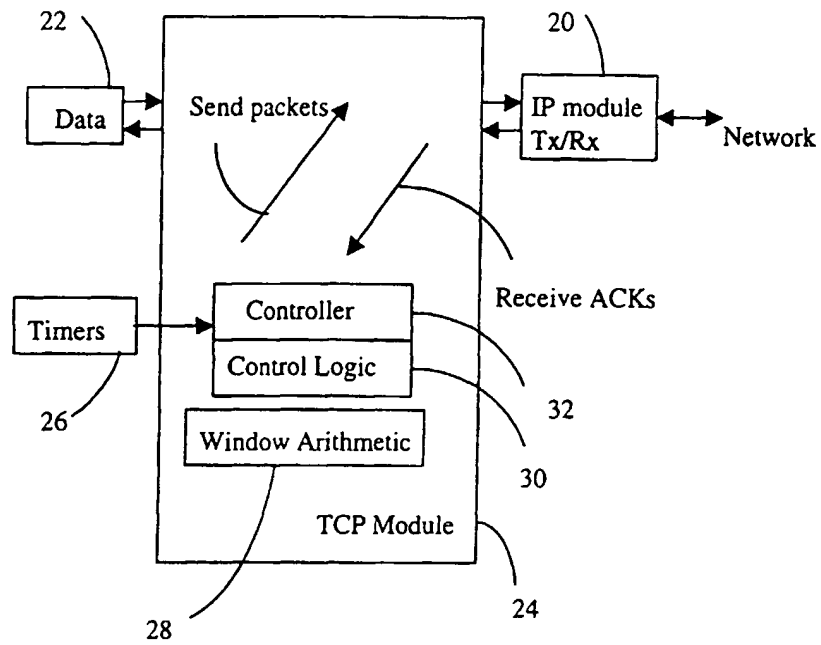


Fig 15

(12) **EUROPEAN PATENT APPLICATION**

(88) Date of publication A3:
10.05.2000 Bulletin 2000/19

(51) Int Cl.7: **H04L 29/08, H04L 29/06,
H04L 12/56**

(43) Date of publication A2:
05.04.2000 Bulletin 2000/14

(21) Application number: **99307651.2**

(22) Date of filing: **28.09.1999**

(84) Designated Contracting States:
**AT BE CH CY DE DK ES FI FR GB GR IE IT LI LU
MC NL PT SE**
Designated Extension States:
AL LT LV MK RO SI

(72) Inventors:

- **Chapman, Alan Stanley John**
Kanata, Ontario K2K 1V5 (CA)
- **Kung, Hsiang-Tsung**
Lexington, MA 02173 (US)

(30) Priority: **30.09.1998 CA 2249152**

(74) Representative: **Dearling, Bruce Clive et al**
Hepworth Lawrence Bryer & Bizley,
Merlin House,
Falconry Court,
Bakers Lane
Epping, Essex CM16 5DQ (GB)

(71) Applicant: **Nortel Networks Corporation**
Montreal, Quebec H2Y 3Y4 (CA)

(54) **Apparatus for and method of managing bandwidth for a packet based connection**

(57) Flow control of packet based traffic by window is known. Novel modification is described which causes the flow control mechanism to reduce sending rate to some configured number rather than just reducing it by a fixed amount such as one half. The description also shows how the flow control mechanism can be constrained to a maximum rate. The configured numbers will assure that the connection can always run at a minimum rate but not more than a maximum rate. If the

guaranteed minimum bandwidth is known and the round trip time between the end points is known or has been calculated, then the sender node needs only reduce its window to that which corresponds to a sending rate equal to that configured number. In this way the protocol will still probe for extra, opportunistic bandwidth but will be able to maintain the minimum rate. In a similar way a window that corresponds to the maximum rate can be calculated and used to constrain the maximum rate of sending.

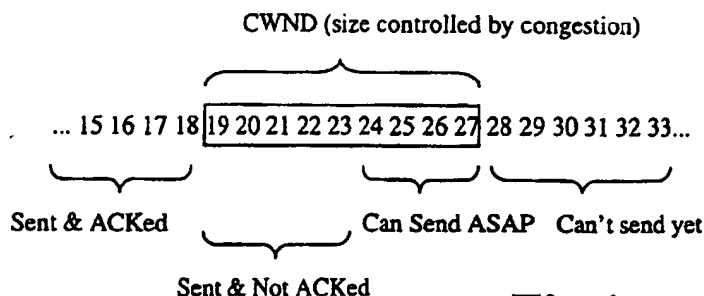


Fig 1

EP 0 991 244 A3



European Patent
Office

EUROPEAN SEARCH REPORT

Application Number
EP 99 30 7651

DOCUMENTS CONSIDERED TO BE RELEVANT			
Category	Citation of document with indication, where appropriate, of relevant passages	Relevant to claim	CLASSIFICATION OF THE APPLICATION (Int.Cl.7)
A	EP 0 458 033 A (IBM) 27 November 1991 (1991-11-27) * page 5, line 53 - page 6, line 40 * * claims 1-9 *	1-14	H04L29/08 H04L29/06 H04L12/56
A	BRAKMO L S ET AL: "TCP VEGAS: NEW TECHNIQUES FOR CONGESTION DETECTION AND AVOIDANCE" COMPUTER COMMUNICATIONS REVIEW,US,ASSOCIATION FOR COMPUTING MACHINERY, NEW YORK, vol. 24, no. 4, 1 October 1994 (1994-10-01), pages 24-35, XP000477041 ISSN: 0146-4833 * page 26, right-hand column, paragraph 3.1 - page 30, right-hand column, line 26 *	1-14	
A	"DYNAMIC COMPUTATION OF TCP MAXIMUM WINDOW SIZE FOR DIRECTLY CONNECTED HOSTS" IBM TECHNICAL DISCLOSURE BULLETIN,US,IBM CORP. NEW YORK, vol. 37, no. 4A, 1 April 1994 (1994-04-01), pages 601-607, XP000446797 ISSN: 0018-8689 * page 601, line 1 - page 604, line 44 *	1-14	TECHNICAL FIELDS SEARCHED (Int.Cl.7) H04L
The present search report has been drawn up for all claims			
Place of search THE HAGUE		Date of completion of the search 20 March 2000	Examiner Karavassilis, N
<p>CATEGORY OF CITED DOCUMENTS</p> <p>X : particularly relevant if taken alone Y : particularly relevant if combined with another document of the same category A : technological background O : non-written disclosure P : intermediate document</p> <p>T : theory or principle underlying the invention E : earlier patent document, but published on, or after the filing date D : document cited in the application L : document cited for other reasons & : member of the same patent family, corresponding document</p>			

EPO FORM 1503 03/02 (Pdc01)

This annex lists the patent family members relating to the patent documents cited in the above-mentioned European search report. The members are as contained in the European Patent Office EDP file on The European Patent Office is in no way liable for these particulars which are merely given for the purpose of information.

Patent document cited in search report	Publication date	Patent family member(s)	Publication date
EP 0458033 A	27-11-1991	US 5063562 A	05-11-1991
		DE 69126640 D	31-07-1997
		DE 69126640 T	08-01-1998
		JP 2783469 B	06-08-1998
		JP 7074780 A	17-03-1995

EPO FORM PO459

Page 845 of 974

APPLICATION NO. 09/912,636

(12) INTERNATIONAL APPLICATION PUBLISHED UNDER THE PATENT COOPERATION TREATY (PCT)

(19) World Intellectual Property Organization
International Bureau



(43) International Publication Date
3 May 2001 (03.05.2001)

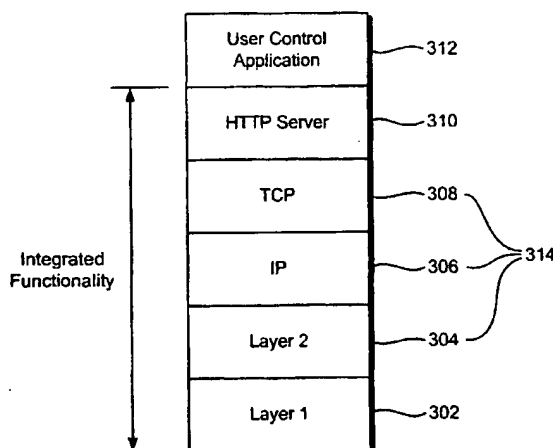
PCT

(10) International Publication Number
WO 01/31852 A1

- (51) International Patent Classification⁷: **H04L 12/28** Southampton, Hampshire SO52 9FT (GB). **KEOGH, David, Bryan** [GB/GB]; 25 Chichester Close, Hedge End, Southampton, Hampshire SO30 2GQ (GB).
- (21) International Application Number: PCT/GB00/03979
- (22) International Filing Date: 18 October 2000 (18.10.2000) (74) Agent: **ALLEN, Derek**; Intellectual Property Department, Siemens Shared Services Limited, Siemens House, Oldbury, Bracknell, Berkshire RG12 8FZ (GB).
- (25) Filing Language: English
- (26) Publication Language: English (81) Designated States (national): CA, JP, US.
- (30) Priority Data:
- | | | |
|-----------|------------------------------|----|
| 9925003.7 | 22 October 1999 (22.10.1999) | GB |
| 9925004.5 | 22 October 1999 (22.10.1999) | GB |
| 0006487.3 | 18 March 2000 (18.03.2000) | GB |
- (84) Designated States (regional): European patent (AT, BE, CH, CY, DE, DK, ES, FI, FR, GB, GR, IE, IT, LU, MC, NL, PT, SE).
- Published:
- With international search report.
 - Before the expiration of the time limit for amending the claims and to be republished in the event of receipt of amendments.
- (71) Applicant (for all designated States except US): **ROKE MANOR RESEARCH LIMITED** [GB/GB]; Roke Manor, Old Salisbury Lane, Romsey, Hampshire SO51 0ZN (GB).
- (72) Inventors; and
- (75) Inventors/Applicants (for US only): **BURNETT, Alan, Mark** [GB/GB]; 6 Cedar Crescent, North Baddesley,

For two-letter codes and other abbreviations, refer to the "Guidance Notes on Codes and Abbreviations" appearing at the beginning of each regular issue of the PCT Gazette.

(54) Title: A FULLY INTEGRATED WEB ACTIVATED CONTROL AND MONITORING DEVICE



(57) Abstract: A web-enabled microcontroller device is provided with both web server functions (310) and generic control and monitoring functions (312). The web-enabled microcontroller device may be embedded in domestic, commercial and industrial hardware. Integrated software for remotely controlling hardware by means of the microcontroller device combines control application code (312) and HTTP server code (310). One implementation of the microcontroller device has a microprocessor coupled to a physical communications unit, a ROM and a RAM. The protocol stack (314) associated with the HTTP server may be permanently coded into the ROM or loaded into the RAM as required. In another implementation, the physical communications unit includes a digital signal processor and a wireless access unit, thereby providing a web-enabled digital wireless access device. In the wireless implementation, the processing of the wireless access physical layer (302) is performed by the digital signal processor and the higher layer processing (310, 312, 314) is performed on the microprocessor.

WO 01/31852 A1

A FULLY INTEGRATED WEB ACTIVATED CONTROL AND MONITORING DEVICE

5 This invention relates to a fully integrated, Web activated control
and monitoring device. The control device of the invention can be
used for remotely controlling and monitoring functions in a physical
device.

10 The Internet and World Wide Web are now ubiquitous
technologies for information transfer. One prominent data networking
standard for information transfer is TCP/IP (Transmission Control
Protocol / Internet Protocol). The majority of home computers are now
supplied 'Internet Ready' and 'Web-enabled', ensuring that many
consumers are not only aware of the technology, but already have it in
their homes.

15 Web servers deliver content according to a given protocol for
presentation on a remote client device, using a Web browser. In the
original World Wide Web (WWW) concept, the given protocol was a
Hypertext Transfer Protocol (HTTP). An HTTP server delivers
Hypertext Markup Language (HTML) pages to one or more users, for
20 display by an HTTP browser. The HTML pages are static files
containing text and image information. This concept has evolved to
include dynamic HTML, whereby interactive pages (having radio
buttons and drop down menus for example) provide an interface to
other applications such as search engines, databases or online
25 dictionaries. Web servers have evolved so that in addition to
supporting HTTP, the Web servers can support other information
transfer protocols including a wireless application protocol (WAP), the

WAP being particularly appropriate for delivering content in wireless implementations.

By embedding a Web server in a target physical device, the WWW concept can be further extended to include the control and monitoring of the physical device. Examples of suitable target physical devices include video recorders, central heating systems and home security systems. The device being controlled can present a protocol-compliant interface to any network using IP (for example the public Internet, an Intranet or a home network) and can thus interface with any Web browser. The structure and content of the protocol-compliant interface presented to the Web browser is stored entirely on the controlled device.

Consider the case of an HTTP server which can be linked to a control application. The control application in turn interfaces with hardware components of the physical device. Examples of hardware components which can be controlled in this manner include: Digital to Analogue Converters (DAC); Analogue to Digital Converter (ADC); parallel Input/Output (I/O) or serial I/O devices; display devices; and device monitoring logic. Rather than simply serving fixed HTML pages, the HTTP server may deliver control and monitoring information using dynamic Web page graphical constructs, for example radio buttons or pull-down menus. Consequently, the control application provides the HTTP server with device status information in HTML format and HTML formatted menus for the setting of device control parameters. The HTTP content presented to a user may be a simulation (or exact copy) of the interface presented by the physical device being controlled.

In general, Web servers have been designed to operate on high performance, general-purpose computing platforms, and are optimised for serving content pages at high speed to a large number of users. It is known to provide HTTP servers for control applications by embedding
5 a general-purpose computer in the physical device to be controlled or by controlling a number of local devices from a single computer. However, such an approach is clearly unsuitable for consumer applications due to the high cost of implementation.

Alternatively physical devices can be controlled by embedded
10 microprocessors (or microcontrollers) rather than general-purpose computers; Web servers suitable for the embedded microprocessors have been produced. Typically, these Web servers are not fundamentally different in design to those used in larger scale general-purpose computing environments. Generally, the Web server software
15 for the embedded microprocessors is smaller and supports fewer features. However, Web servers for general purpose computers and servers for the embedded microprocessors do still have the following features in common.

Firstly, both types of Web server are implemented as a separate
20 piece of code that executes in a separate thread or process alongside the control application code.

Secondly, an external scheduling system (usually part of an operating system) determines when both the Web server software and the control application software get execution time.

25 Thirdly, as the control application code and the Web server code are both designed to execute separately, code needed to perform standard tasks will often be duplicated in both, adding to the overall size of the software.

Lastly, both types of Web server code incorporate code which handles differences between protocols. In the case of the HTTP, there is no single standard definition, rather there are multiple standard versions, each of which has many optional features. In addition, there
5 are a number of non-HTTP features that are widely considered to be server options. Since a Web server is typically a separate piece of code, designed to be applied across a wide range of applications, the Web server usually contains code to support functionality that is not used by a given specific application. The implementation of a Web
10 server in an embedded environment as described above is not optimal in a typical consumer appliance implementation using a low cost microcontroller. The memory and processing power available in typical consumer appliances are very restricted.

Ideally, the Web server software for implementation in
15 embedded environments should include no redundant code and the control application software should be able to dictate when any support software, such as the Web server software, receives execution time. When this control application software is implemented in embedded microcontroller devices, the embedded microcontroller devices may be
20 used to replace existing, known microcontroller devices. In addition, the cost and inconvenience of wiring is a significant barrier to home networking. The implementation of a wireless environment is thus desirable and so there is a need for Web server technology in the context of digital radio devices in order to produce a class of physical
25 device particularly appropriate for home networking implementations.

According to the present invention, there is provided a microcontroller device for controlling at least one physical device, the microcontroller device including a microprocessor coupled to a

physical communication unit and a memory unit, wherein the memory unit stores an integrated piece of software arranged to perform a server function and a control application function and to support protocol stacks.

5 Preferably, the physical communication unit includes a UART chip. Equally preferably, the physical communication unit includes a digital signal processor and a wireless access unit. More preferably the digital signal processor is a baseband processor.

10 Preferably, the microcontroller device is embedded in a single integrated circuit.

 Preferably the memory unit includes at least one non-volatile memory unit, the memory unit advantageously, permanently storing the integrated piece of software. More preferably, the non-volatile memory is read only memory (ROM) unit.

15 Alternatively, the memory unit includes at least one volatile memory unit. The integrated software is preferably stored as an image on the at least one volatile memory unit. The at least one volatile memory unit may be a random access memory (RAM) unit.

20 Preferably, the server functions provided by the integrated piece of software comply with a communications protocol. More preferably, the communications protocol is a Hypertext Transfer Protocol and protocol stacks supported by the integrated piece of software are compatible with the Hypertext Transfer Protocol. Equally preferably, the communications protocol is a Wireless Application Protocol and
25 protocol stacks supported by the integrated piece of software are compatible with the Wireless Application Protocol.

 Preferably, there is provided a physical device controlled by the microcontroller device. More preferably, there is provided a

communications network comprising at least one physical device controlled by the microcontroller device.

According to the present invention, there is also provided a method of remotely controlling a physical device having at least one function, the method including the steps of: providing the physical device with a server engine for receiving incoming communications from a remote user; providing the physical device with a command handler arranged to produce an interpreted remote user communication by interpreting the incoming communication; providing the physical device with at least one control application arranged to receive the interpreted remote user communication and to interface with the physical device for controlling the at least one function; and controlling the at least one function in response to the at least one control application.

Preferably, a memory unit having an integrated piece of software recorded thereon is provided, wherein the integrated piece of software performs the method above.

At least one embodiment of the present invention will now be described, by way of example only, with reference to the following drawings, in which:

Figure 1 is a schematic diagram of an IP-based home network;

Figure 2A is a schematic diagram of the layout of a first web-enabled microcontroller device;

Figure 2B is a schematic diagram of the layout of a second web-enabled microcontroller device for use in a wireless network environment;

Figure 3 is a first reference model for the first web-enabled generic microcontroller device of Figure 2A;

Figure 4 is a schematic diagram of a network showing wireless and wired inter-working between devices using different physical layer media; and

Figure 5 is a schematic diagram of the integration of an HTTP server with a control application.

Throughout the following description, identical reference numerals are used to identify like parts.

Referring to Figure 1, an IP based home network 102 is coupled to a plurality of physical devices, the IP based home network 102 being in communication with the public Internet 100 via an IP gateway 104. The plurality of physical devices can include: computing devices, such as a home computer 114 or a Personal Digital Assistant (PDA) 128; home entertainment devices, such as a television 116, a video recorder 112, home audio equipment 120; domestic appliances, such as a refrigerator 124, a fax machine 126, a mobile telephone 130, a microwave oven 108 or heating and lighting facilities 118; security equipment 110; a utility meter 106, such as a gas meter or an electricity meter; and automotive monitoring equipment such as car engine management equipment 122.

Remote control of the plurality of physical devices requires that each physical device has a microcontroller arrangement for control and monitoring of the physical device, and embedded HTTP server software. The HTTP server software permits communication with a management terminal, for example the home computer 114 or the wireless web enabled PDA 128. The plurality of physical devices are managed using an HTTP browser implemented upon the management terminal in order to present information generated by the HTTP server software in the form of content pages. In order to manage the plurality

of physical devices, the management terminal requires respective IP addresses associated with each of the plurality of physical devices. The respective IP addresses can be obtained by an automatic discovery process, for example the Bluetooth 'Service Discovery Protocol'.

5 Although reference has been made to "Bluetooth" it should be appreciated that other wireless access techniques, for example, Infrared Data Association (IrDA) or Shared Wireless Access Protocol (SWAP), can be used to reduce the cost and complexity of network wiring.

10 In a first embodiment of the invention (Figure 2A), a web-enabled microcontroller arrangement 200 includes a physical communications unit 202 coupled to a microprocessor 210 via a first data bus 212, a non-volatile memory unit 204 (e.g. ROM, EEPROM, EPROM or flash memory) and a volatile memory unit 206 (e.g. RAM or random access flash) also being coupled to the first data bus 212.

15 The microcontroller arrangement 200 is suitable for connection to a network for communications with remote devices coupled to the network by means of the physical communications unit 202, for example, a Universal Asynchronous Receiver/Transmitter chip (UART).

20 Referring to Figure 3, the web-enabled microcontroller arrangement 200 implements an integrated protocol stack 314 providing a data link layer (Layer 2) 304, an IP layer 306 and a TCP layer 308, the IP layer 306 and the TCP layer 308 providing transmission and networking layers. The integrated protocol stack 314

25 is overlaid upon a physical layer 302 (Layer 1). An HTTP layer 310 is overlaid upon the TCP layer 308, the HTTP layer 310 being implemented by HTTP server software. The HTTP server software, the integrated protocol stack 314 and user control application software

312 all execute on the embedded microprocessor 210. The physical layer 302, the data link layer 304, the IP layer 306, the TCP layer 308 and the HTTP layer 310 are either coded permanently into the non-volatile memory unit 204 (for example, mask programmed onto ROM) or provided as an image which may be loaded by a software developer into the volatile memory unit 206. In the latter case, the software developer is free to generate control application software specific to the hardware of the controlled device and linking directly to the HTTP server software.

10 The code build process incorporates the building of both control application code and HTTP server code to give a single piece of web-enabled software. This approach gives a number of advantages in terms of overall size and complexity of the web-enabled software, and in terms of utilisation of available processing power.

15 Firstly, the HTTP server software presents a generic user interface with content dependent only on the controlled device itself. The HTTP server software uses a networking standard (TCP/IP) which allows a device to be monitored or controlled from anywhere in the world with no translation whatsoever of the content presented by the HTTP server. The device can be controlled from a known platform; for example, a workstation, the home personal computer 114 or the PDA 128.

20 Secondly, the HTTP server code and the control application code are combined to form a single piece of software in which the HTTP server code only executes when called by the control application code. By structuring the HTTP server code so that it periodically returns thread mastery to the control application code throughout the processing of an incoming HTTP communication, and by providing

parameters to configure this processing both dynamically and as part of the code build, the application is given a high degree of control over the available processing power. When active, the HTTP server can be seen as making use of spare capacity not used for the control
5 application.

Thirdly, since the HTTP server code is part of an implementation specific code build process, the web-enabled control application software includes code to support any and all desirable HTTP server features. By the use of standard code building techniques and suitable
10 structuring of the HTTP server code, any HTTP server code supporting functionality that is optional, yet not relevant for the specific implementation for the physical device, can simply be excluded from the final software as part of the code build process for that implementation.

15 Finally, as the control application code and HTTP server code form part of the same piece of web-enabled software, they can both make use of the same sections of code to perform standard tasks rather than duplicate these sections in each case.

In a second embodiment of the present invention (Figure 2B),
20 the microcontroller arrangement 200 differs from that of the first embodiment (Figure 2A) by the physical communication unit 202 having an embedded DSP 214 and a radio frequency (RF) transceiver device 216, for transmitting and receiving signals.

In the context of the reference model of Figure 3, the physical
25 layer 302 is supported by the transceiver unit 216 and the embedded DSP 214. Signals received by the transceiver unit 216 are processed by the embedded DSP 214, the embedded DSP 214 can also support the (baseband) data link layer 304 (layer 2). Although specific

apparatus have been described for the full or partial support of layers 1 and 2 of the reference model, it should be appreciated that other hardware known in the art can be used to support the layers 1 and 2.

Some functionality of the data link layer 304, the IP layer 306, the TCP layer 308 and the HTTP layer 210 is provided by the embedded microprocessor 210. Typical wireless access technologies that use this device architecture include: Digital Enhanced Cordless Telecommunications (DECT), Global System for Mobile communications (GSM), Bluetooth, Universal Mobile Telecommunications System (UMTS), IrDA or SWAP.

It should be understood that the embedded microprocessor 210 can be instructed to perform the operations of the DSP 214 in addition to Layer 2 and control processing, thus the DSP 214 is not absolutely essential. Alternatively, logic circuitry can be provided to perform some or all tasks of the DSP 214 and the embedded microprocessor 210.

Referring to Figure 4, the wired and wireless inter-working between devices of Figure 1 can be seen in more detail. Layers 1 and 2 of the wired part of the network differ from layers 1 and 2 of the wireless part of the network. However, the IP layer 306, the TCP layer 308 and the HTTP server layer 310 are identical, making inter-working straightforward.

In addition to the above described applications, physical devices comprising the microcontroller arrangement 200 can be used for diagnostic and configuration functions by a remote service centre. For example, a user can have difficulty configuring one of the physical devices or the physical device may appear to be faulty. The user can connect the physical device to the remote service centre via the public

Internet 100, where skilled service personnel can run diagnostics remotely or assist the user in the configuration of the physical device. Additionally, the home entertainment equipment incorporating the embedded Web server can include specialised configuration and diagnostic functions not accessible to the user. The configuration and diagnostic functions, inaccessible to the user, can be accessed by a remote customer service centre via the public Internet 100.

The home security systems 110 can also have additional remote management features. A security company can use embedded HTTP server technology to provide additional management services to a client. The HTTP server software can be used to control and monitor audio and video surveillance equipment, and also to control building infrastructure functions, for example, heat and power systems.

In another application, the utility meter 106 can be read and controlled remotely by both a customer and a utility provider via the wired public Internet 100 or a wireless network in order to save on collection/reading costs and to provide a common user interface.

The HTTP server software embedded in the car engine management system 122 can provide a valuable diagnostic tool. The engine management system 122 collects performance and service interval information, which can be interrogated locally by a garage, or remotely over the public Internet 100 by a manufacturer's service centre or a breakdown service. Information gathered can be used for a number of purposes including: by the garage to identify items requiring adjustment/replacement during service; by the garage to identify a fault after breakdown; by the breakdown service to identify a fault before dispatching roadside assistance; and by the manufacturer to gather performance information over the life of a vehicle.

In order to exclude unauthorised users from the home IP network 102 connected to the public Internet 100 where authorised users are relatively unskilled, a firewalling technique is employed. In general, the firewalling technique restricts access to local networks from the unauthorised users. The firewalling technique is implemented at the gateway 104 between the public Internet 100 and the home IP network 102. Access to the home IP network 102 is restricted by applying security measures known in the art including filtering of incoming packets on combinations of identification numbers including source IP addresses, destination IP addresses, TCP/IP port numbers and UDP (User Datagram Protocol) port numbers.

In a wired network the firewalling function is performed by an IP access router known in the art acting as a Local Area Network (LAN) switch/router for the controlled devices connected to it. However, in a wireless network, it is possible that the gateway is only an access point, and performs no switching or routing functions for the devices in communication with the wireless network. The firewalling technique is still required for the wireless network, but no physical media switching is needed. Consequently, the gateway 104 comprises a firewalling device implementing the firewalling technique and the protocol relay function in order to provide an access point to the public Internet 100 to a home IP network 102. In this case, the firewalling device is just another web-enabled physical device and can be managed, configured and diagnosed remotely, with expert help as required.

Referring to Figure 5, the control application 506 is capable of relaying commands to a hardware control module 530 and receives responses from the hardware control module 530. The HTTP server

software includes an HTTP server engine 502 and a plurality of command handlers 504. Each command handler 504 is a piece of application specific code that the HTTP server engine 502 uses to handle a respective application specific aspect of an HTTP communication for the control application 506.

The control application 506 registers each command handler 504 with the HTTP server engine 502. The HTTP server engine 502 then calls one of the plurality of command handler 504 appropriate for handling a respective application specific aspect of an HTTP communication at various stages of the HTTP communication. The registration process associates a textual name with each command handler 504, thus allowing the appropriate command handler 504 to be specified within the HTTP communication (not shown). The HTTP server engine 502 looks for textual names of command handlers 504 within HTTP communications and maps each textual name found, using the list of registered command handlers, to an appropriate corresponding command handler 504.

Furthermore, the command handlers 504 are specific to the hardware ultimately being controlled through the hardware control module 530. Any parameters and context specific content passed by the HTTP server engine 502 to the control application 506 via a given command handler 504 are interpreted by the given command handler 504, the command handler 504 acting as an interface between the control application 506 and the Web server engine 502. In response to the parameters and context specific content delivered via the command handler 504, the control application 506 performs whatever action is appropriate given the nature of the HTTP communication (not shown). The command handlers 504 also interpret the results of actions by the

control application 506 and act as the interface back to the HTTP server engine 502, for example the command handlers 504 identify errors and/or produce parameters and context specific content for the HTTP server engine 502 to format into an outgoing HTTP communication (not shown). The HTTP server engine 502 handles all aspects of the HTTP connection, from receiving and sending of HTTP communications, to parsing and formatting the parameters and context specific content.

The HTTP server engine 502 also handles sequencing, by calling an appropriate command handler 504 at various points throughout the process of handling an HTTP communication. The sequencing is, however, still under the overall control of the control application 506, due to other features of the HTTP server software. The control application 506 can, either as part of the build process or dynamically for each HTTP communication, specify parameters, for instance, how much data may be sent or received over the TCP/IP connection before returning execution to the control application 506. Additionally, in the sequence of events that are required to handle an HTTP communication, there are fixed points where the control application 506 can define whether the HTTP server engine 502 continues automatically with the next event of the sequence or returns execution to the control application 506 after each event. The HTTP server engine 502 therefore is only allowed execution time when called by the control application 506. The amount of execution time allocated at each call to the HTTP server engine 502 can thus be configured, as can the number of simultaneous HTTP communications that can be handled.

The user can still interact locally with the hardware of the physical device through a local user interface 540. Local user communications (not shown) are handled directly by the control application 506, the control application 506 acting as an interface to the hardware control module 530. Feedback from the hardware control module 530 returns to the user interface 540 via the control application 506.

It will be understood that, although the foregoing description is concerned with HTTP server code, other protocols can be adopted in place of hypertext transfer protocol. Most notably the Wireless Application Protocol (WAP) is suitable for the implementation of a Web server in a wireless environment. Suitable alternative protocols within which the invention can be applied include file transfer protocol (FTP), Session Initiation Protocol (SIP), Service Location Protocol (SLP), telnet and secure HTTP (SHTTP). Suitable protocol stacks for the physical layer 302 and the data link layer (Layer 2) 304 include: X.25, AppleTalk, Ethernet and asynchronous transfer mode (ATM).

Claims:

1. A microcontroller device for controlling at least one physical device, the microcontroller device including a microprocessor coupled to a physical communication unit and a memory unit, wherein the memory unit stores an integrated piece of software arranged to perform a server function and a control application function and to support protocol stacks.
2. A microcontroller device according to Claim 1, wherein the physical communication unit includes a UART chip.
3. A microcontroller device according to Claim 1, wherein the physical communication unit includes a digital signal processor and a wireless access unit.
4. A microcontroller device according to Claim 3, wherein the digital signal processor is a baseband processor.
5. A microcontroller device according to Claims 1, 2, 3 or 4, wherein the microcontroller device is embedded in a single integrated circuit.
6. A microcontroller device according to any one of Claims 1 to 5, wherein the memory unit includes at least one non-volatile memory unit.

7. A microcontroller device according to Claim 6, wherein the at least one non-volatile memory unit permanently stores the integrated piece of software.

5 8. A microcontroller device according to Claims 6 or 7, wherein the at least one non-volatile memory unit is a read only memory unit.

9. A microcontroller device according to any one of Claims 1 to 6, wherein the memory unit includes at least one volatile memory unit.

10

10. A microcontroller device according to Claim 9, wherein the integrated piece of software is stored as an image on the at least one volatile memory unit.

15 11. A microcontroller device according to Claims 9 or 10, wherein the at least one volatile memory unit is a random access memory unit.

12. A microcontroller device according to any one of the preceding claims, wherein the server functions provided by the integrated piece of software comply with a communications protocol.

20

13. A microcontroller device according to Claim 12, wherein the communications protocol is a Hypertext Transfer Protocol and protocol stacks supported by the integrated piece of software are compatible with the Hypertext Transfer Protocol.

25

14. A microcontroller device according to Claim 12, wherein the communications protocol is a Wireless Application Protocol and

protocol stacks supported by the integrated piece of software are compatible with the Wireless Application Protocol.

5 15. A physical device controlled by a microcontroller device as claimed in any one of the preceding claims.

16. A communications network comprising at least one physical device according to Claim 15.

10 17. A method of remotely controlling a physical device having at least one function, the method including the steps of:

providing the physical device with a server engine for receiving incoming communications from a remote user;

15 providing the physical device with a command handler arranged to produce an interpreted remote user communication by interpreting the incoming communication;

providing the physical device with at least one control application arranged to receive the interpreted remote user communication and to interface with the physical device for controlling the at least one function; and

20

controlling the at least one function in response to the at least one control application.

25 18. A memory unit having an integrated piece of software recorded thereon, wherein the integrated piece of software performs the method as claimed in Claim 17.

19. A microcontroller device substantially as hereinbefore described with reference to the accompanying drawings.

20. A network of microcontroller devices substantially as
5 hereinbefore described with reference to the accompanying drawings.

21. A physical device substantially as hereinbefore described with reference to the accompanying drawings.

1/5

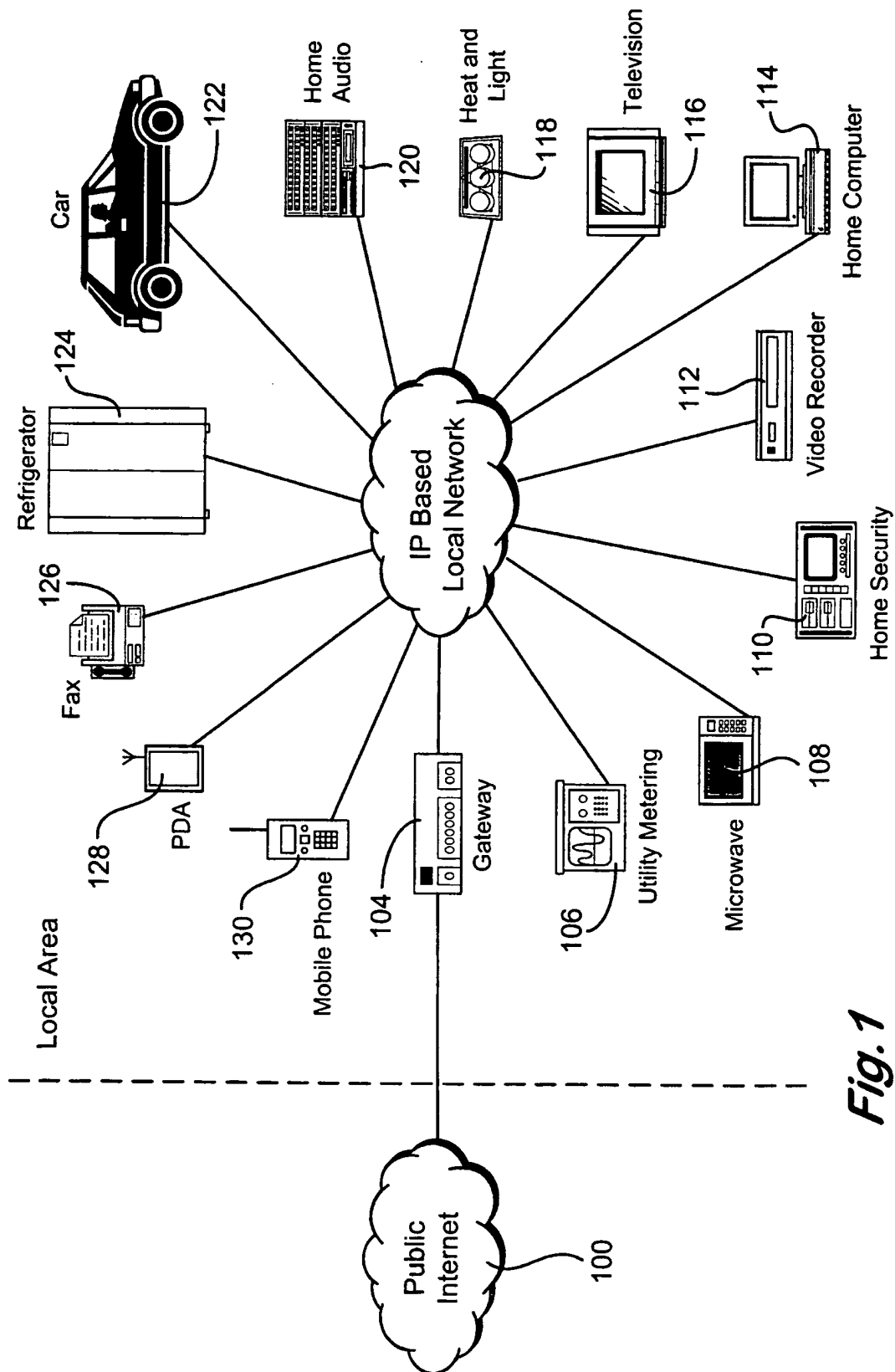
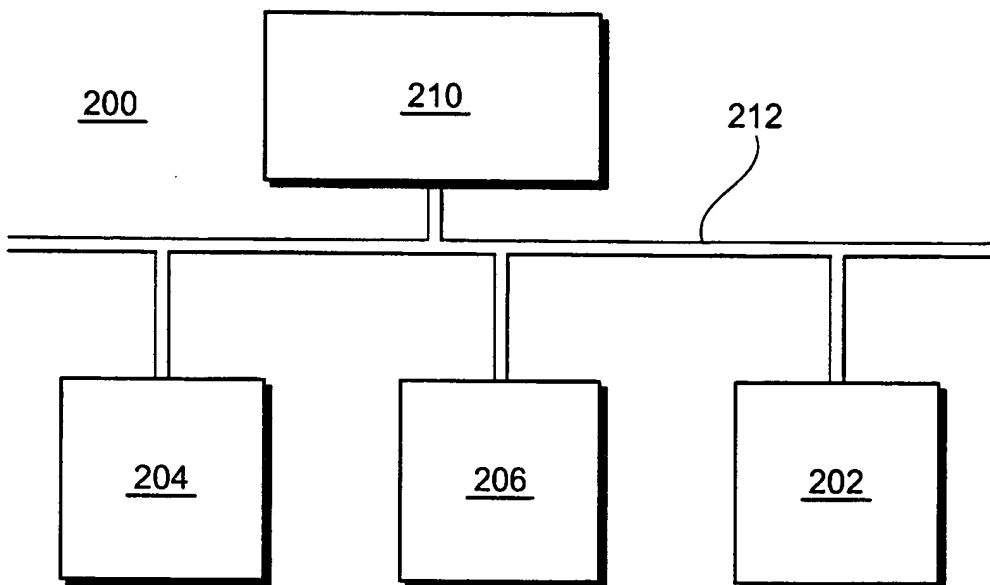
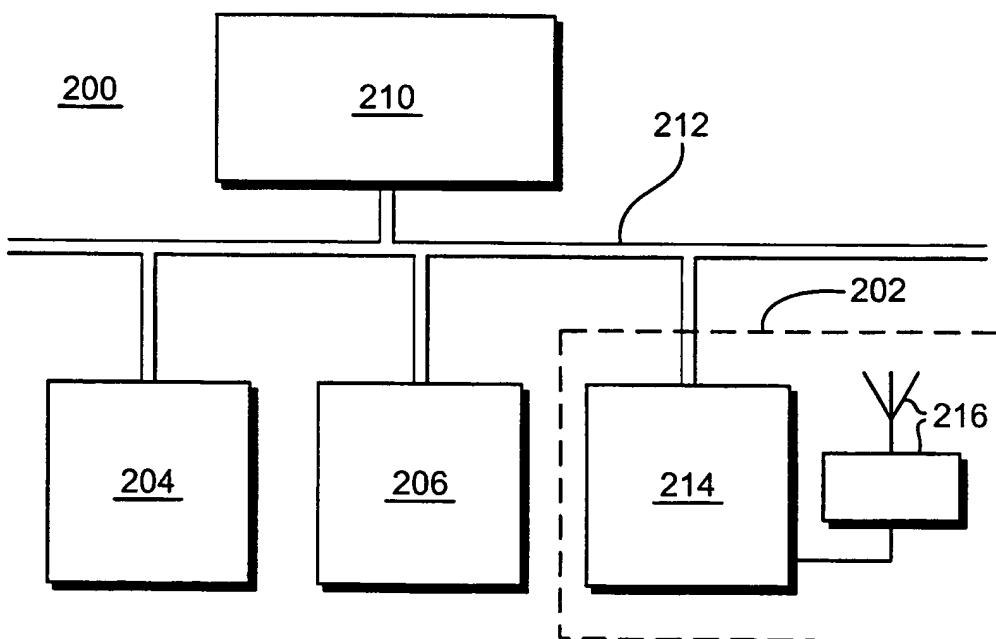
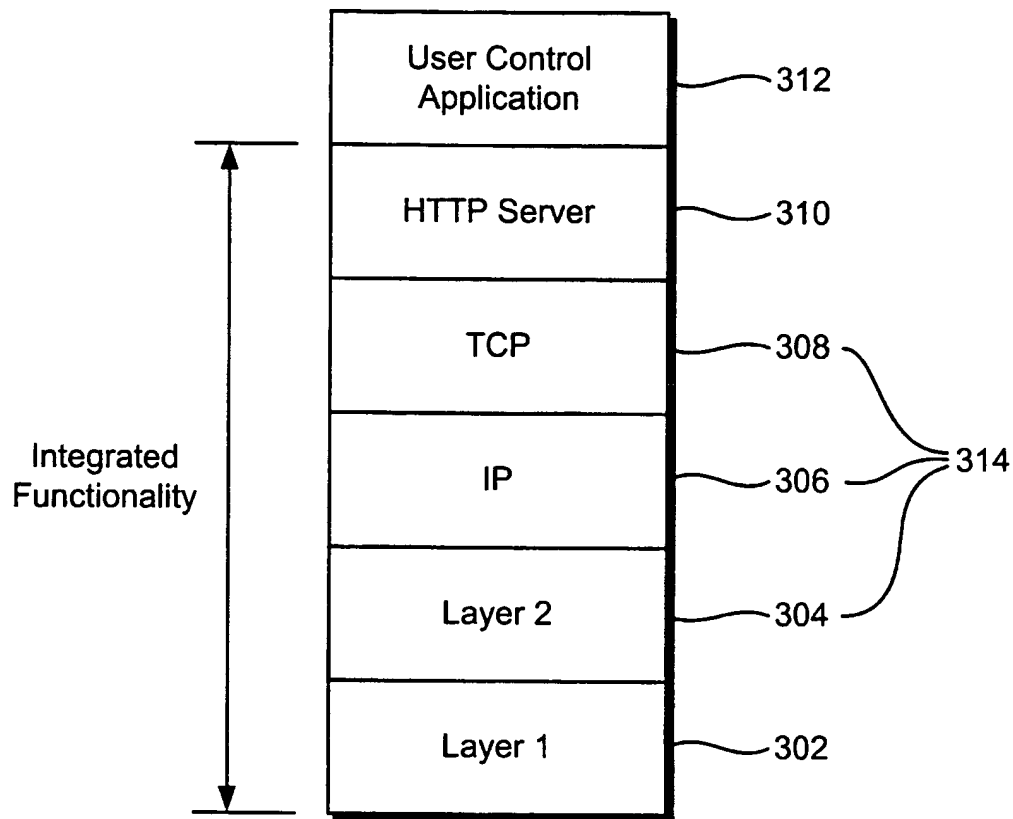


Fig. 1

2/5

*Fig. 2A**Fig. 2B*

3/5

*Fig. 3*

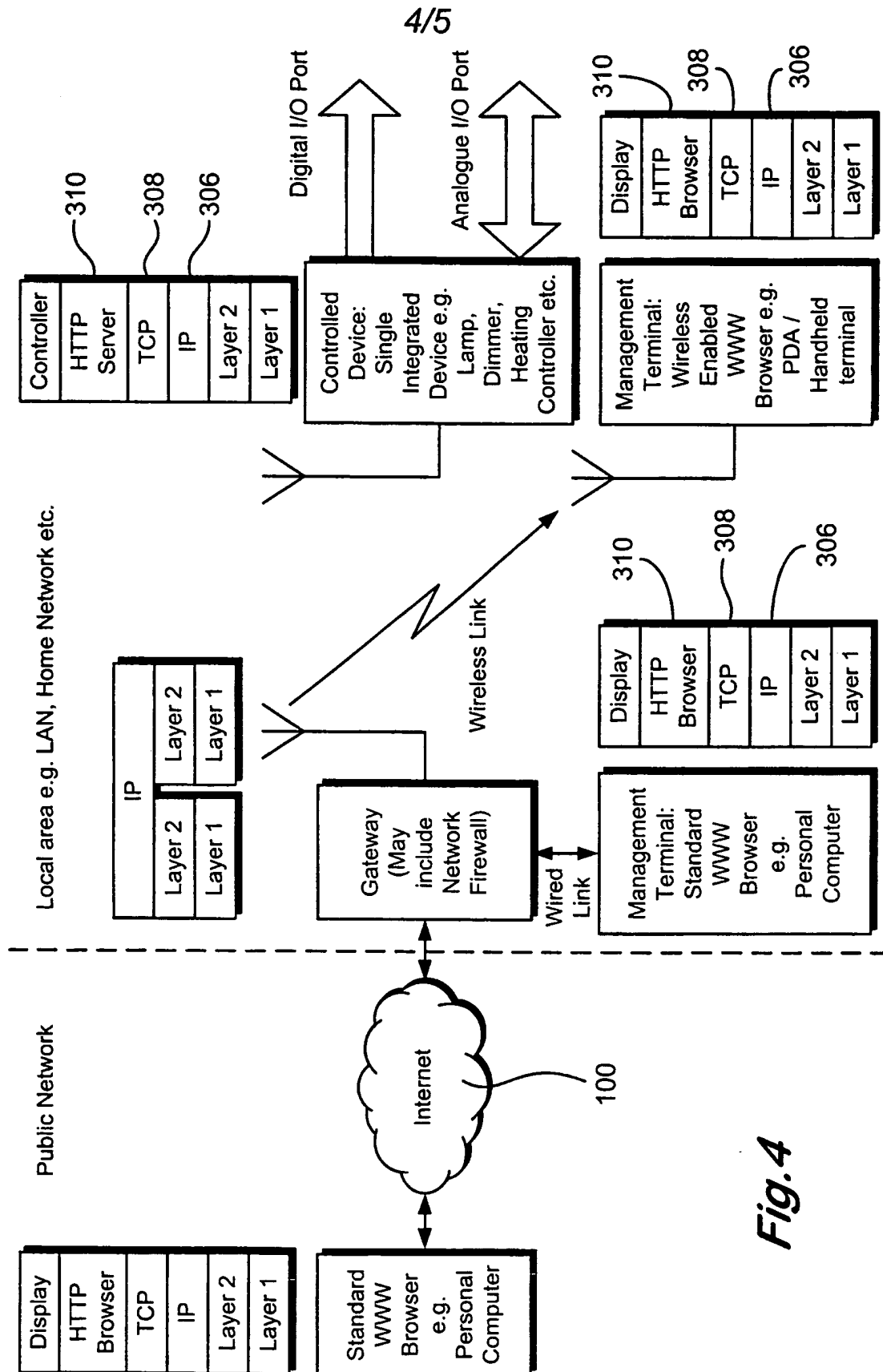


Fig. 4

5/5

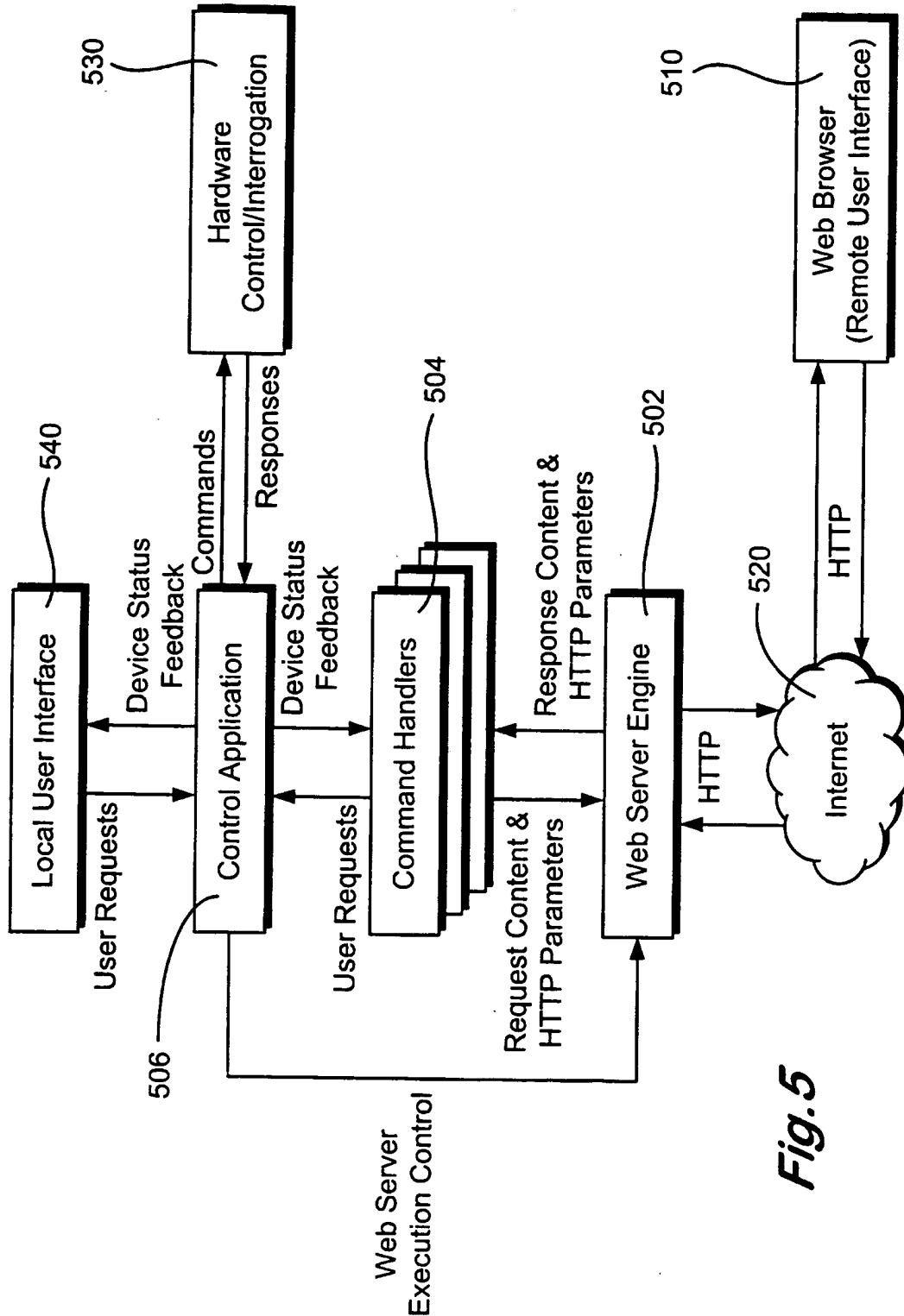


Fig. 5

INTERNATIONAL SEARCH REPORT

Inte: nal Application No
PCT/GB 00/03979

A. CLASSIFICATION OF SUBJECT MATTER

IPC 7 H04L12/28

According to International Patent Classification (IPC) or to both national classification and IPC

B. FIELDS SEARCHED

Minimum documentation searched (classification system followed by classification symbols)

IPC 7 H04L

Documentation searched other than minimum documentation to the extent that such documents are included in the fields searched

Electronic data base consulted during the international search (name of data base and, where practical, search terms used)

EPO-Internal

C. DOCUMENTS CONSIDERED TO BE RELEVANT

Category *	Citation of document, with indication, where appropriate, of the relevant passages	Relevant to claim No.
X	<p>PEISEL B: "DESIGNING THE NEXT STEP IN INTERNET APPLIANCES" ELECTRONIC DESIGN,US,PENTON PUBLISHING, CLEVELAND, OH, vol. 46, no. 7, 23 March 1998 (1998-03-23), page 50,52,56 XP000780455 ISSN: 0013-4872 page 50, left-hand column, line 1 -page 56, left-hane column, line 58 --- -/--</p>	<p>1,5-8, 12,13, 15-21</p>

☒ Further documents are listed in the continuation of box C.

☒ Patent family members are listed in annex.

* Special categories of cited documents :

A document defining the general state of the art which is not considered to be of particular relevance

E earlier document but published on or after the international filing date

L document which may throw doubts on priority claim(s) or which is cited to establish the publication date of another citation or other special reason (as specified)

O document referring to an oral disclosure, use, exhibition or other means

P document published prior to the international filing date but later than the priority date claimed

T later document published after the international filing date or priority date and not in conflict with the application but cited to understand the principle or theory underlying the invention

X document of particular relevance; the claimed invention cannot be considered novel or cannot be considered to involve an inventive step when the document is taken alone

Y document of particular relevance; the claimed invention cannot be considered to involve an inventive step when the document is combined with one or more other such documents, such combination being obvious to a person skilled in the art.

Z document member of the same patent family

Date of the actual completion of the international search

28 February 2001

Date of mailing of the international search report

06/03/2001

Name and mailing address of the ISA

European Patent Office, P.B. 5818 Patentlaan 2
NL - 2280 HV Rijswijk
Tel. (+31-70) 340-2040, Tx. 31 651 epo nl,
Fax: (+31-70) 340-3016

Authorized officer

Pham, P

INTERNATIONAL SEARCH REPORT

International Application No
PCT/GB 00/03979

C.(Continuation) DOCUMENTS CONSIDERED TO BE RELEVANT

Category *	Citation of document, with indication, where appropriate, of the relevant passages	Relevant to claim No.
X	LAWTON G: "DAWN OF THE INTERNET APPLIANCE" COMPUTER,US,IEEE COMPUTER SOCIETY, LONG BEACH., CA, US, vol. 30, no. 10, 1 October 1997 (1997-10-01), page 16,18 XP000738076 ISSN: 0018-9162 the whole document -----	1,9, 11-13, 15-21
X A	WO 99 46746 A (ABB POWER T & D CO) 16 September 1999 (1999-09-16) page 3, line 24 -page 7, line 3 page 8, line 20 -page 9, line 3 -----	1,12,13, 15-21 3

INTERNATIONAL SEARCH REPORT

Information on patent family members

Inte. nal Application No

PCT/GB 00/03979

Patent document cited in search report	Publication date	Patent family member(s)	Publication date
WO 9946746 A	16-09-1999	AU 6453498 A	27-09-1999
		EP 1062648 A	27-12-2000
		NO 20004486 A	08-11-2000

PCT

WORLD INTELLECTUAL PROPERTY ORGANIZATION
International Bureau



INTERNATIONAL APPLICATION PUBLISHED UNDER THE PATENT COOPERATION TREATY (PCT)

(51) International Patent Classification 7 : H04L 29/00		A2	(11) International Publication Number: WO 00/67443 (43) International Publication Date: 9 November 2000 (09.11.00)
(21) International Application Number: PCT/NO00/00144 (22) International Filing Date: 28 April 2000 (28.04.00) (30) Priority Data: 19992125 30 April 1999 (30.04.99) NO (71) Applicant (for all designated States except US): TELEFONAK- TIEBOLAGET LM ERICSSON [SE/SE]; S-126 25 Stock- holm (SE). (72) Inventor; and (75) Inventor/Applicant (for US only): NILSEN, Børge [NO/NO]; Lørenveien 34, N-0585 Oslo (NO). (74) Agent: OSLO PATENTKONTOR AS; Postboks 7007 M, N-0306 Oslo (NO).		(81) Designated States: AE, AG, AL, AM, AT, AT (Utility model), AU, AZ, BA, BB, BG, BR, BY, CA, CH, CN, CR, CU, CZ, CZ (Utility model), DE, DE (Utility model), DK, DK (Utility model), DM, DZ, EE, EE (Utility model), ES, FI, FI (Utility model), GB, GD, GE, GH, GM, HR, HU, ID, IL, IN, IS, JP, KE, KG, KP, KR, KR (Utility model), KZ, LC, LK, LR, LS, LT, LU, LV, MA, MD, MG, MK, MN, MW, MX, NO, NZ, PL, PT, RO, RU, SD, SE, SG, SI, SK, SK (Utility model), SL, TJ, TM, TR, TT, TZ, UA, UG, US, UZ, VN, YU, ZA, ZW, ARIPO patent (GH, GM, KE, LS, MW, SD, SL, SZ, TZ, UG, ZW), Eurasian patent (AM, AZ, BY, KG, KZ, MD, RU, TJ, TM), European patent (AT, BE, CH, CY, DE, DK, ES, FI, FR, GB, GR, IE, IT, LU, MC, NL, PT, SE), OAPI patent (BF, BJ, CF, CG, CI, CM, GA, GN, GW, ML, MR, NE, SN, TD, TG). Published <i>Without international search report and to be republished upon receipt of that report.</i>	
(54) Title: ADAPTION OF SERVICES IN A TELEPHONE NETWORK			
(57) Abstract The present invention relates to an arrangement for improving the service architecture for a compound network, comprising several types of access, as well as comprising parallel service nodes/networks for respective access technologies, and for the purpose of making customer specific adaptations to the service layer more flexible and allowing for a more cost-effective support of access specific protocols and service, it is according to the present invention suggested that said arrangement comprises an open service control protocol allowing support of access specific protocols and services while also allowing the respective access networks to share the same access nodes and service architectures.			

FOR THE PURPOSES OF INFORMATION ONLY

Codes used to identify States party to the PCT on the front pages of pamphlets publishing international applications under the PCT.

AL	Albania	ES	Spain	LS	Lesotho	SI	Slovenia
AM	Armenia	FI	Finland	LT	Lithuania	SK	Slovakia
AT	Austria	FR	France	LU	Luxembourg	SN	Senegal
AU	Australia	GA	Gabon	LV	Latvia	SZ	Swaziland
AZ	Azerbaijan	GB	United Kingdom	MC	Monaco	TD	Chad
BA	Bosnia and Herzegovina	GE	Georgia	MD	Republic of Moldova	TG	Togo
BB	Barbados	GH	Ghana	MG	Madagascar	TJ	Tajikistan
BE	Belgium	GN	Guinea	MK	The former Yugoslav Republic of Macedonia	TM	Turkmenistan
BF	Burkina Faso	GR	Greece	ML	Mali	TR	Turkey
BG	Bulgaria	HU	Hungary	MN	Mongolia	TT	Trinidad and Tobago
BJ	Benin	IE	Ireland	MR	Mauritania	UA	Ukraine
BR	Brazil	IL	Israel	MW	Malawi	UG	Uganda
BY	Belarus	IS	Iceland	MX	Mexico	US	United States of America
CA	Canada	IT	Italy	NE	Niger	UZ	Uzbekistan
CF	Central African Republic	JP	Japan	NL	Netherlands	VN	Viet Nam
CG	Congo	KE	Kenya	NO	Norway	YU	Yugoslavia
CH	Switzerland	KG	Kyrgyzstan	NZ	New Zealand	ZW	Zimbabwe
CI	Côte d'Ivoire	KP	Democratic People's Republic of Korea	PL	Poland		
CM	Cameroon	KR	Republic of Korea	PT	Portugal		
CN	China	KZ	Kazakstan	RO	Romania		
CU	Cuba	LC	Saint Lucia	RU	Russian Federation		
CZ	Czech Republic	LI	Liechtenstein	SD	Sudan		
DE	Germany	LK	Sri Lanka	SE	Sweden		
DK	Denmark	LR	Liberia	SG	Singapore		
EE	Estonia						

ADAPTION OF SERVICES IN A TELEPHONE NETWORK**FIELD OF THE INVENTION**

- 5 The present invention relates to an arrangement for improving the service architecture for a compound telephone network, comprising several types of access/protocols as well as comprising parallel service nodes.
- 10 In other words, the present invention finds its application in the field of H.323 and service control.

BACKGROUND OF THE INVENTION

- 15 The present invention has been developed in connection with problems encountered when making service specific adaptations to the requirements of the individual customer in a network.
- 20 The problem is that the respective service logic is highly integrated into the switching logic of the core network. This means that new service logic and service adaptations may require changes in the core switching functions. This again makes it very hard to make customer specific adaptations to the service layer.
- 25

For example, big changes in the IN services can not be made without updating the switches in the network, i.e. even if the control in reality is between the handset and node 1 in the network, there is "transport equipment" that also has to be updated as service control uses the same control data/mechanisms/paths as these transport nodes.

30

KNOWN SOLUTIONS AND PROBLEMS WITH THESE

35

As explained above the main problem with traditional networks is that the service control protocols are integrated with lower layer functions such as call and media control.

This binds the service control plane to lower layer functions such as basic switching functions and introduce system couplings that make customisation of service control expensive.

5

The service control and services are thus often tied to given access protocols. This often leads to building of parallel service networks with minor differences in protocols and services. Each service network then solves the
10 service issues for one specific type of net. This typically results in costly and ineffective service development frameworks that are inflexible, costly and hard to maintain.

15 For the IP telephone networks as defined by the H.323 standard, the service control protocol is defined by the H.450 standards-suite. This defines an in-band service control protocol that is carried within the H.225.0 call control plane (a defined subset of Q.931). This protocol defines a
20 set of ASN.1 service control messages that are used for invoking and controlling services. The problems with this approach is summarized as:

25 Introduction of new services requires updates of H.450 messages and decoding logic in the gatekeeper. This slows down introduction of service logic as it requires both a standardisation process and updates to the switch control plane.

30 Adaptation of service control to vendor specific control messages/logic becomes impossible (or costly) as it relates to the switching core.

Integration and interworking with messaging protocols becomes heavier as it requires more transcoding of messages.
35 The user-user messages and user-service messages are carried within the same messages and framing. In order to

identify user-service messages, all messages need to be analysed - not only those addressed for services. The protocol is ASN.1 encoded and does not easily integrate with MIME encoded messaging services.

5

OBJECTS OF THE INVENTION

An object of the present invention is to provide an arrangement, by which a cost effective and adaptive service architecture for a compound network comprising several types of access, can be implemented in a far more expedient and versatile manner.

Another object of the present invention is to provide an arrangement, by which the service networks can more easily be integrated and developed.

Still another object of the present invention is to provide an arrangement whereby the service architecture and access technologies are made more flexible and more easy to maintain.

BRIEF SUMMARY OF THE INVENTION

The above objects are achieved in an arrangement as stated in the preamble, which according to the present invention is characterized in that said comprises an open service control protocol allowing support of access specific protocols and services while also allowing the respective access networks to share the same access nodes and service architectures, said open service control protocol is adapted for removing the coupling between the access/service technology and the switching logic in the core network.

In other words, the invention aims at customization of service control protocols by allowing the service control to be specialised independently from the other control functions such as call control and media control.

The invention proposes to use an open service control protocol that allows for a more cost-effective support of access specific protocols and services while also allowing
5 the respective access networks to share the same service nodes and service architectures. The solution also aims at removing the coupling between the access/service technology and the switching logic in the core network. The proposed solution is based on H.323 being extended with HTTP for the
10 service control.

Further features and advantages will appear from the following description taken in conjunction with the enclosed drawings, as well as from the enclosed patent claims.

15

BRIEF DISCLOSURES OF THE DRAWINGS

Figure 1 is a schematical layout illustrating multiple access type and service node architecture.

20

Figure 2 is a schematical layout illustrating the general principle of the present invention, comprising composed single service node with plugin architecture, giving homogeneous architecture with access adapters.

25

Figure 3 is a simplified block diagram illustrating a reference model according to the present invention.

Figure 4 is a schematical layout illustrating a service
30 node structure according to an embodiment of the present invention.

Figure 5 is a schematical layout illustrating an example of usage according to the present invention.

35

DETAILED DESCRIPTION OF EMBODIMENTS

In Figure 1 there is in a schematical layout illustrated a prior art multiple access type and service node architecture.
5

In order to elevate the problems related to the service architecture for such a compound network comprising several types of access, as well as comprising parallel service
10 nodes/networks for respective access technologies, it is according to the present invention suggested to use an open service control protocol, as this will be discussed in further detail with reference to Figure 2.

15 Figure 2 illustrates a homogeneous service architecture with access adapters, according to the present invention, by which the open service control protocol can be implemented, so as to allow for a more cost effective support of access specific protocols and services while also allowing
20 active access networks to share the same service nodes and service architectures.

With the suggested solution which can be embedded in the general layout according to Figure 2, it is also possible
25 to remove the coupling between the access/service technology and the switching logic in the core network.

The proposed solution is based on H.323 being extended with HTTP for service control.

30 More specifically, the proposed solution replaces the H.450 standards suites with the more open and extendable HTTP protocol. The solution also makes use of the feature set of HTTP to add the required flexibility. Among these features
35 are found:

HTTP Tunnelling

The tunnelling feature refers to the use of the HTTP transport layer protocol for carrying other data protocols (in which case the HTTP headers carry information about what kind of payload type/protocol that is being carried).

Server Side Plugin

The plugin approach represents the server side equivalent of browser plugins where 3rd party plugins (objects/functions) can be added dynamically. The invocation of a plugin is controlled through the content-type field or through selections/filters on the given path. The binding between the plugin and the invocation criteria is set through configuration.

Servlet Functions

The servlet approach relates to servlet objects that implements CGI like functions, but may add persistency over sessions and is being object oriented.

DESCRIPTION OF SOLUTION

The invention disclosure relates to an H.323 based telephone network where clients makes phone calls through a central call-/control processing switch called a gatekeeper. The gatekeeper performs call-/control processing functions such as charging, routing and resource control and may also activate call related services on a service node according to the call states and the user profiles. When the client and the gatekeeper talks different languages/dialects, an access node is added in between to perform the required gateway functions.

In Figure 3 there is illustrated a simplified network reference model.

This invention disclosure proposes to replace the H.450 based service control protocol between the access nodes and the gatekeeper/service node with an HTTP based protocol.

5 This means that service configuration- and control messages are being HTTP encoded by the access nodes and decoded, analysed and executed by the service node. This again means that there is no normalisation of the service protocols in the access nodes and that the mapping from the access spe-

10 cific service data to the service node languages are being performed by service node plugins/servlets.

The service node represents a set of software processes being capable of executing phone services and interacting

15 with the gatekeeper. The service node thus provides a set of service functions and offers a programming API for service execution and control. The service node does also provide an HTTP server that supports HTTP tunnelling, servlets and server side plugins. Through the use of this HTTP

20 server it is possible to write a plugin or servlet that interacts with the programming API in order to control the service execution. An example of this could be a plugin that translates DTMF codes to API method calls, e.g. DTMF '*23*1*1530#' may translate to API method 'userIs-

25 BusyTo(15.30)'.

In Figure 4 there is given an illustration of the service node architecture.

30 In order to provide acceptable service availability the service node and the HTTP server will need to support installations of new plugins/servlets in runtime. Further, the architecture needs to be such that faulty plugins, servlets or sessions does not impair the operation of the

35 service node.

The described architecture and feature set supports access specific service control messages as well as customer spe-

cific adaptation of the service network and the service control protocols, though within the limits of the feature set of the service API. This is illustrated through the following two examples.

5

In order to add an access specific service control protocol such as QSIG, the vendor would need to write an access node and a service plugin/servlet.

10 The QSIG access-node would translate the call- and connection control messages into the H.323 format, but would tunnel the QSIG messages inside HTTP messages and address these to the service node.

15 A QSIG plugin/servlet would be written and installed on the HTTP server of the service node. The logic of this plugin/servlet would translate the QSIG messages into method calls (and capability sets) in the service API. When a QSIG service control message is sent from a PBX, the
20 access node will wrap the QSIG message into an HTTP frame and send it to the service node. The HTTP server on the service node will receive the package, detect that the format is something called QSIG, look up in its configuration data and activate the correct plugin/servlet for QSIG. This
25 plugin/servlet will analyse the QSIG message, make method calls in the API and return the appropriate QSIG encoded response.

When new features are added to the service node and the
30 service API, the updates can be provided to the access specific parts through updates of the plugins/servlets, i.e. there are no updates required in the access nodes.

In order to add a provider specific service control protocol e.g. based on GSM-SMS, the provider would need to write
35 a plugin/servlet that translates the GSM-SMS messages into method calls over the service API. The procedures are as defined above, but in this case an external 3rd part can do

provider specific customisation to the service network without being tied up to new deliveries of the core system. In Figure 5 it is illustrated the GSM-SMS example alongside a default option.

5

ADVANTAGES

Added Flexibility

- 10 The service control protocol is more flexible in terms of supporting different service control data formats/encoding standards. For each new encoding standard, a new plugin needs to be encoded.

Simplicity

- The service control protocol becomes more flexible in that it is simpler to add new service control messages and supporting these. It becomes simpler to debug the system, to
20 secure the message transport (cf. SSL) and to get the data through firewalls and proxies.

Customisation

- 25 The solution allows the service provider to add provider specific service control messages independent from the system solution provider. This means that a provider can add new control messages for decoding these independent from the system provider (e.g. add a new SMS message and update
30 the plugin for decoding this).

Performance

- The messages are being addressed towards the correct recipient, meaning that the gatekeeper does not need to analyse all messages (incl. user-user msg.) in order to pick
35 up the user-service data.

BROADENING**1) Integration with Messaging Applications**

5 The HTTP service control format follows the MIME encoding standard that is used by SMTP, NTP and S/MIME messaging applications. It is expected that it should be possible to integrate this service control with these messaging applications.

10

2) Support for Notification Services

The principle can be extended to allow the application server to issue HTTP messages/notifications to the clients (e.g. when the client registers). This can for example be used for notifying the user about new e-mail messages in the in-box.

15

3) Extensions for Supporting the SIP Protocol

20

The SIP protocol builds on using the HTTP protocol and can probably be integrated into the system solution relatively simple if the application server supports call-from-the-blue services.

25

4) Terminal (gateway) to Terminal (gateway) Service Control

If two terminals (or their gateways) want to exchange service control/data they could exchange this service control/data on a language that they have agreed on. The respective entities (terminals or gateways) can also dynamically download transcoder servlets/plugins from a central depository upon need.

30

35 This could for example be used when user A on his PC is sending user B on a GSM terminal an email message. The GSM gateway decides that email is not understood and retrieves some transcoder for handling this email. The choice of

transcoder can be selected according to user preferences, previous user behaviour, network or operator criteria. Examples of transcoders here could be:

- 5 • Transcoder from email to GSM-SMS message
- Transcoder from email to voice rendering
- Transcoder from email to WAP

5) Access Control based on Service Control Plugin

10

The access to transcoder functions (servlets/plugins) can be controlled according to subscription profiles, user locations and other metrics of the system. Further more can the invoked transcoder function apply access control on the
15 specific information elements of the service control/data protocols. This could e.g. be used to control when and from where a given service is used and what kind of service data that is legal in the given context. An example could be to filter on the contents of a GSM-SMS message to ensure that
20 no pornographic data is being transmitted. (The transcoder would in this case act as an application layer firewall.)

APPENDIX

Terminology

ITU-H.323	A family of ASN.1 encoded protocols defining message formats, encoding standards and call state sequences of multimedia conferences on an Internet protocol infrastructure.
ITU-H.225.0	A subset of the H.323 standards suite being based on Q.931 and defining call control messages, encoding standards and call-state sequences.
ITU-H.450	A suite of ASN.1 standards defining service control protocols to be used for service control in an H.323 network. The H.450 messages are being carried within H.225.0 messages.
ITU-Q.931	Telephony standard for call control that defines call control messages, encoding standards and call-state sequences.
ASN.1	Abstract Syntax Notation Number 1 A formal data structure definition language
HTTP	A MIME (ascii) encoded protocol for transport of world-wide-web data. The protocol is open for tunnelling of other protocols.
CGI	Common Gateway Interface A script language used for customisation of web page contents
API	Application Programming Interface
DTMF	Dual Tone Multiple Frequency
QSIG	A service control protocol used by PBX
PBX	Private Branch Exchange
GSM	Global System for Mobile Communication A widely employed standard for mobile communication
SMS	Short Message Service Messaging service protocol employed within GSM
SSL	Secure Socket Layer Security protocol employed for Transport Layer

	Security
MIME	Multipart Information Message Entity Protocol encoding format based on ascii characters
SIP	Session Initiation Protocol IP Telephony protocol based on HTTP
SMTP	Simple Mail Transfer Protocol Protocol for transport/exchange of email mes- sages
NTTP	Network News Transfer Protocol Protocol for transport/exchange of news mes- sages
S/MIME	Secure MIME
WAP	Wireless Access Protocol A web protocol for mobile devices (i.e. 'a- kind-of' HTTP for mobile handsets)

P a t e n t c l a i m s

1. Arrangement for providing an improved service architecture for a compound telephone network,
5 c h a r c t e r i z e d i n that said arrangement comprises an open service control protocol allowing support of access specific protocols and services while also allowing the respective access networks to share the same access nodes and service architectures,
10 said open service control protocol is adapted for removing the coupling between the access/service technology and the switching logic in the core network.
2. Arrangement as claimed in claim 1,
15 c h a r c t e r i z e d i n that said open service control protocol is based on H.323 standard for communication across Internet Protocol (IP) based networks, said H.323 standard being extended with HTTP (Hyper Text Transport Protocol) for the service control.
- 20 3. Arrangement as claimed in claim 1 or 2,
c h a r c t e r i z e d i n that said H.323 standard with extended HTTP protocol is adapted to enhance or replace the H.450 suite of protocols, the adaptation also
25 making use of the feature set of HTTP to add the required flexibility.
4. Arrangement as claimed in claim 3,
c h a r c t e r i z e d i n that said features of the
30 HTTP may include:
- HTTP tunnelling
 - Service Side Plugin
 - Servlet Functions
- 35 5. Arrangement as claimed in claim 3 or 4,
c h a r c t e r i z e d i n that between the access nodes and the gatekeeper/service node there is introduced

an HTTP based protocol, which entails that the service configuration and control messages are being HTTP encoded by the access nodes and decoded, analysed and executed by the service node.

5

6. Arrangement as claimed in claim 5,
c h a r c t e r i z e d i n that no normalisation of
the service protocols in the access node has to be performed, and that the mapping from the access specific service data to the service node languages are being performed
10 by service node plugins/servlets.

7. Arrangement as claimed in claim 6,
c h a r c t e r i z e d i n that the service node represents a set of software processes being capable of executing phone services and interacting with the gatekeeper,
15 the service node thus providing a set of service functions and offering a programming API for service execution and control.

20

8. Arrangement as claimed in claim 1,
c h a r c t e r i z e d i n that said network includes a service node which also provides an HTTP server that supports HTTP tunnelling, servlets and server side plugins,
25 the use of this HTTP server making it possible to write a plugin or servlet that interacts with the programming API in order to control the service execution.

9. Arrangement as claimed in claim 8,
30 c h a r c t e r i z e d i n that the arrangement comprises a plugin/servlet that translates from access specific service control to generic service API method calls.

10. Arrangement as claimed in claim 9,
35 c h a r c t e r i z e d i n that the access node is adapted to tunnel the access specific service control data to the plugin/servlet by use of HTTP, which plugin/servlet

then transcodes this access specific service control to said generic service API method calls.

11. Arrangement as claimed in claim 9 or 10,
5 c h a r c t e r i z e d i n that a server side plugin/servlet can be installed and updated in run-time.

12. Arrangement as claimed in claim 11,
c h a r c t e r i z e d i n that said server side
10 plugin/servlet can be provided by 3rd parties.

13. Arrangement as claimed in any of the claims 9-12,
c h a r c t e r i z e d i n that said server automati-
cally selects correct plugin/servlet according to config-
15 ured rules in the server and the type of service control data being signalled, the type of data being signalled is indicated by the HTTP protocol.

14. Arrangement as claimed in any of the claims 9-13,
20 c h a r c t e r i z e d i n that the plugin/servlet will format the return codes/states from the generic API calls to access specific return codes/states and return these using the HTTP protocol.

25 15. Arrangement as claimed in claim 14, c h a r a c t e r i z e d i n that the respective enti- ties involved (terminal or gateway) can dynamically down- load transcoder servlets/plugins from a central repository upon need.

30 16. Arrangement as claimed in claims 15, c h a r a c t e r i z e d i n that access to transcoder functions (servlets/plugins) can be controlled according to subscriber profiles, user locations and other metrics of
35 the system.

17. Arrangement as claimed in claims 15-16,
c h a r a c t e r i z e d i n that the invoked

transcoder function can be arranged to apply access control to the specific information elements of the service control/data protocols.

Fig-1: Access Specific Service Architecture / multiple Service Nodes

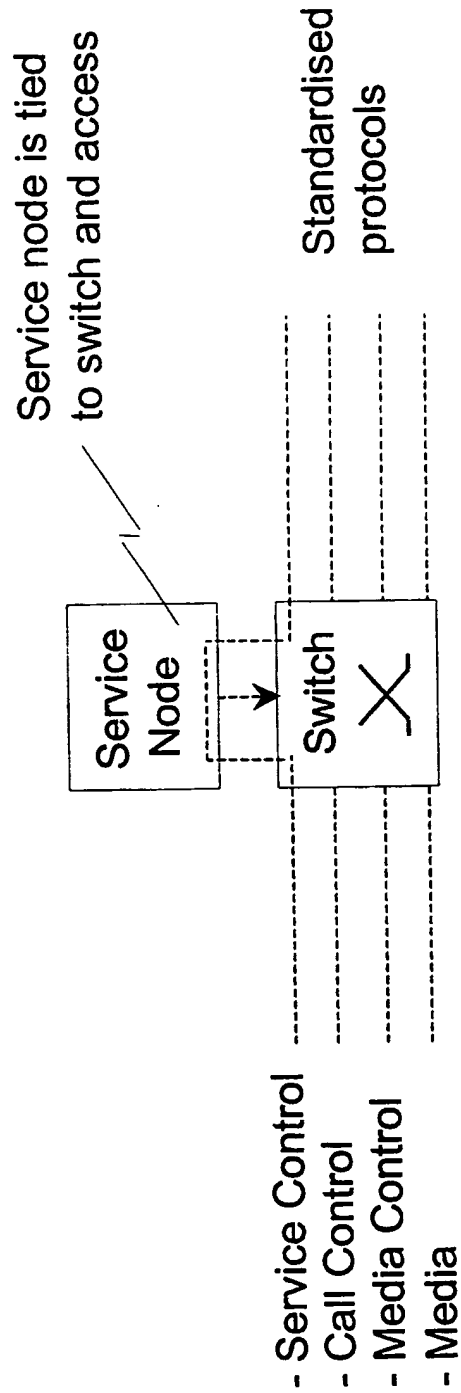


Fig-2: Homogenous Service Architecture with Access Adaptors

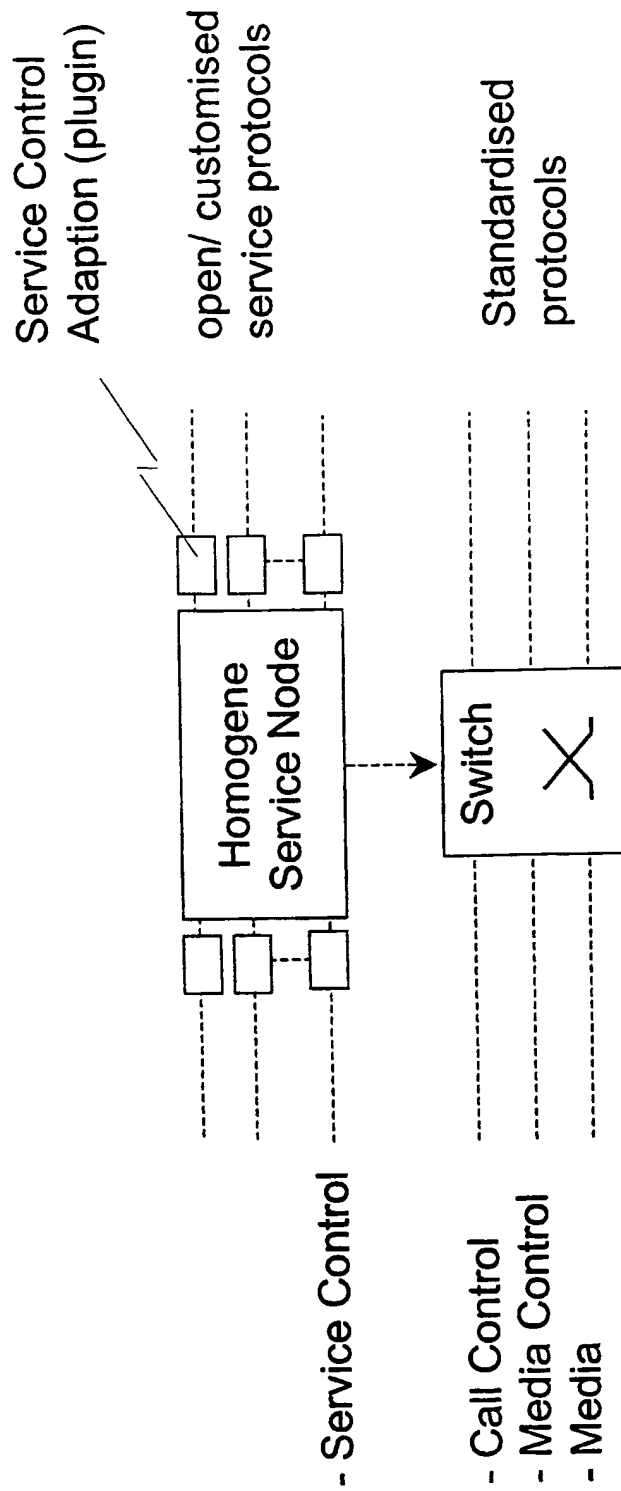


Fig-3: Simplified Reference Model

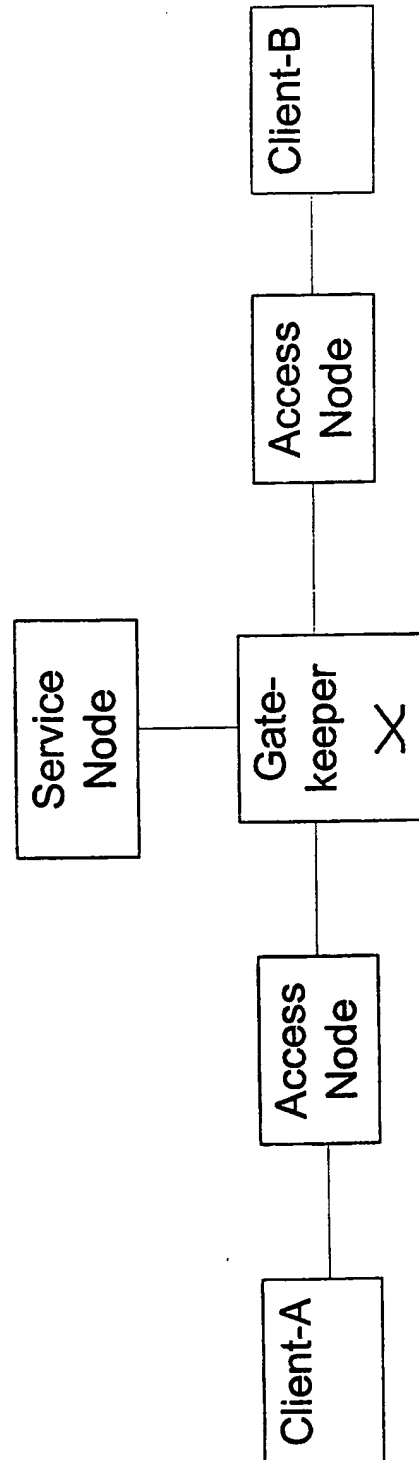
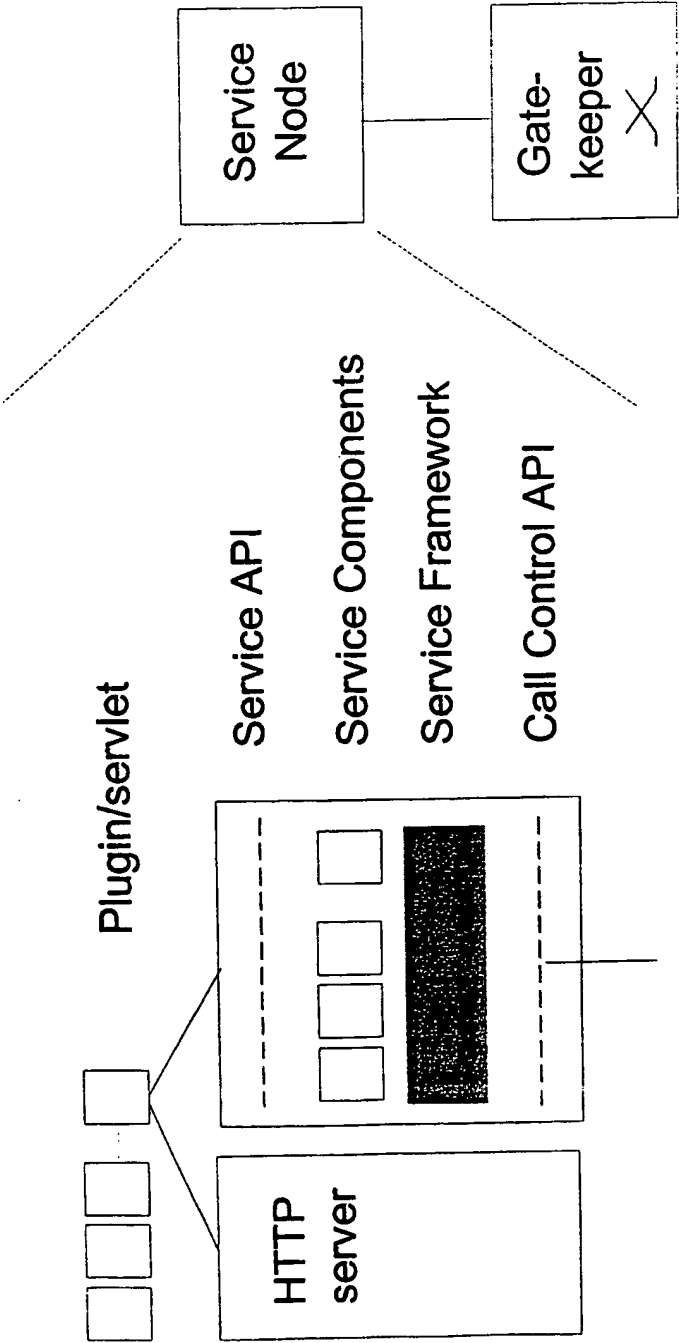
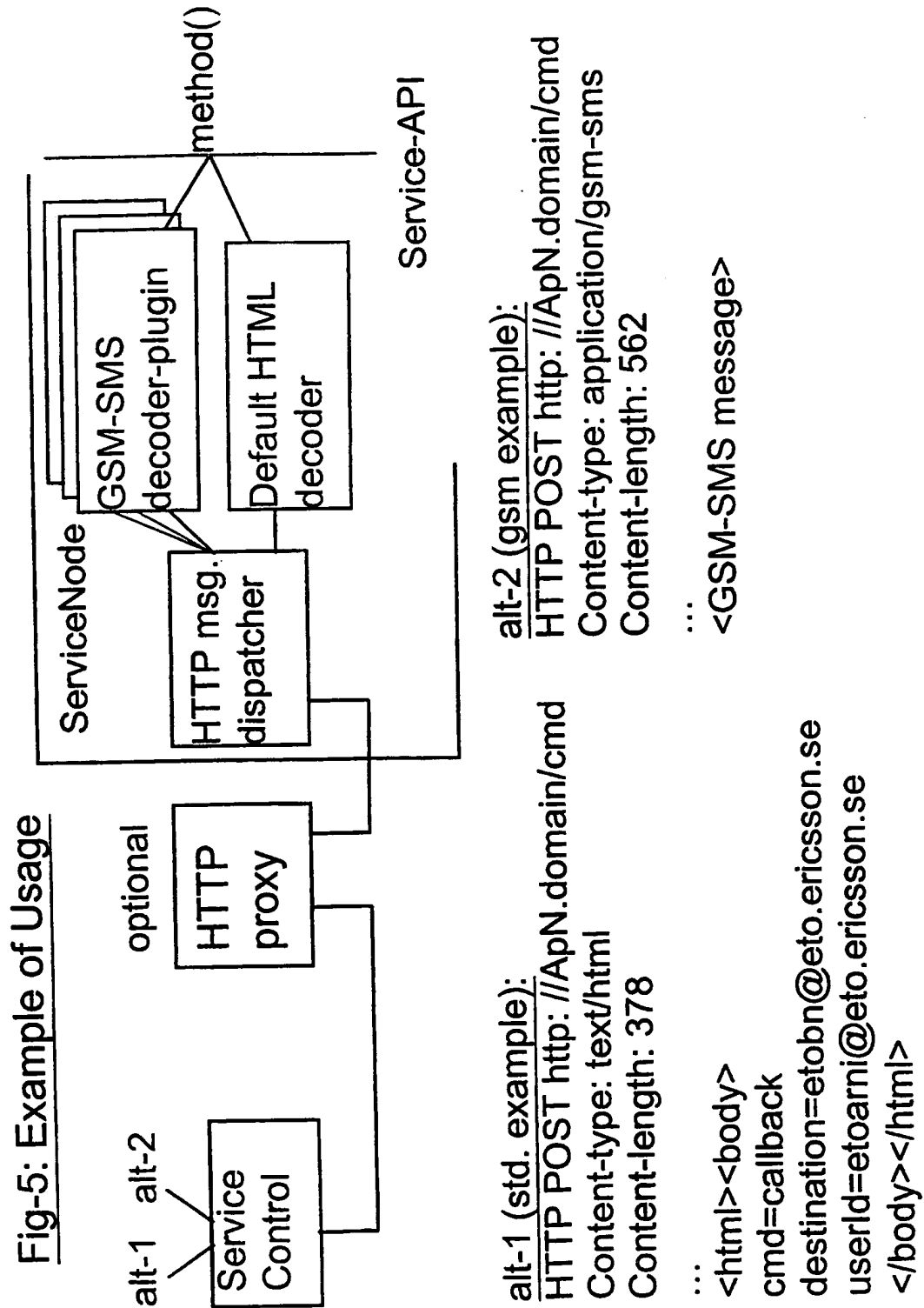


Fig-4: Service Node Structure





(12) INTERNATIONAL APPLICATION PUBLISHED UNDER THE PATENT COOPERATION TREATY (PCT)

(19) World Intellectual Property Organization
International Bureau



(43) International Publication Date
9 November 2000 (09.11.2000)

PCT

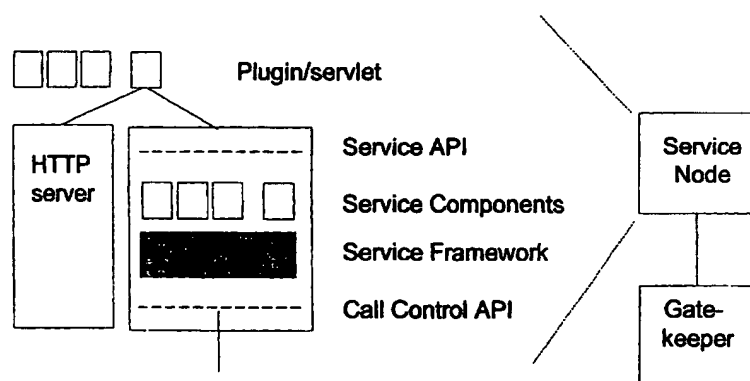
(10) International Publication Number
WO 00/67443 A3

- (51) International Patent Classification⁷: **H04L 12/64** (81) Designated States (*national*): AE, AG, AL, AM, AT, AT (utility model), AU, AZ, BA, BB, BG, BR, BY, CA, CH, CN, CR, CU, CZ, CZ (utility model), DE, DE (utility model), DK, DK (utility model), DM, DZ, EE, EE (utility model), ES, FI, FI (utility model), GB, GD, GE, GH, GM, HR, HU, ID, IL, IN, IS, JP, KE, KG, KP, KR, KR (utility model), KZ, LC, LK, LR, LS, LT, LU, LV, MA, MD, MG, MK, MN, MW, MX, NO, NZ, PL, PT, RO, RU, SD, SE, SG, SI, SK, SK (utility model), SL, TJ, TM, TR, TT, TZ, UA, UG, US, UZ, VN, YU, ZA, ZW.
- (21) International Application Number: PCT/NO00/00144
- (22) International Filing Date: 28 April 2000 (28.04.2000)
- (25) Filing Language: English
- (26) Publication Language: English
- (30) Priority Data:
19992125 30 April 1999 (30.04.1999) NO (84) Designated States (*regional*): ARIPO patent (GH, GM, KE, LS, MW, SD, SL, SZ, TZ, UG, ZW), Eurasian patent (AM, AZ, BY, KG, KZ, MD, RU, TJ, TM), European patent (AT, BE, CH, CY, DE, DK, ES, FI, FR, GB, GR, IE, IT, LU, MC, NL, PT, SE), OAPI patent (BF, BJ, CF, CG, CI, CM, GA, GN, GW, ML, MR, NE, SN, TD, TG).
- (71) Applicant (*for all designated States except US*): TELEFONAKTIEBOLAGET LM ERICSSON [SE/SE]; S-126 25 Stockholm (SE).
- (72) Inventor; and
- (75) Inventor/Applicant (*for US only*): NILSEN, Børge [NO/NO]; Lørenveien 34, N-0585 Oslo (NO). Published:
— With international search report.
- (74) Agent: OSLO PATENTKONTOR AS; Postboks 7007 M, N-0306 Oslo (NO). (88) Date of publication of the international search report:
25 January 2001

[Continued on next page]

(54) Title: ADAPTION OF SERVICES IN A TELEPHONE NETWORK

Service Node Structure



(57) Abstract: The present invention relates to an arrangement for improving the service architecture for a compound network, comprising several types of access, as well as comprising parallel service nodes/networks for respective access technologies, and for the purpose of making customer specific adaptations to the service layer more flexible and allowing for a more cost-effective support of access specific protocols and service, it is according to the present invention suggested that said arrangement comprises an open service control protocol allowing support of access specific protocols and services while also allowing the respective access networks to share the same access nodes and service architectures.



For two-letter codes and other abbreviations, refer to the "Guidance Notes on Codes and Abbreviations" appearing at the beginning of each regular issue of the PCT Gazette.

INTERNATIONAL SEARCH REPORT

International Application No

PCT/NO 00/00144

A. CLASSIFICATION OF SUBJECT MATTER
IPC 7 H04L12/64

According to International Patent Classification (IPC) or to both national classification and IPC

B. FIELDS SEARCHED

Minimum documentation searched (classification system followed by classification symbols)

IPC 7 H04L H04Q H04M

Documentation searched other than minimum documentation to the extent that such documents are included in the fields searched

Electronic data base consulted during the international search (name of data base and, where practical, search terms used)

EPO-Internal

C. DOCUMENTS CONSIDERED TO BE RELEVANT

Category *	Citation of document, with indication, where appropriate, of the relevant passages	Relevant to claim No.
E	EP 1 003 343 A (NORTEL NETWORKS CORP) 24 May 2000 (2000-05-24) the whole document	1
P,A	--- DANIELE RIZZETTO ET AL: "A voice over IP Service Architecture for Integrated Communications" IEEE INTERNET COMPUTING, vol. 3, May 1999 (1999-05) - June 1999 (1999-06), pages 53-62, XP002901209 ISSN 1089-7801 the whole document	1
P,A	--- EP 0 996 270 A (NORTEL NETWORKS CORP) 26 April 2000 (2000-04-26) the whole document --- -/-	1

☒ Further documents are listed in the continuation of box C.

☒ Patent family members are listed in annex.

* Special categories of cited documents :

- *A* document defining the general state of the art which is not considered to be of particular relevance
- *E* earlier document but published on or after the international filing date
- *L* document which may throw doubts on priority claim(s) or which is cited to establish the publication date of another citation or other special reason (as specified)
- *O* document referring to an oral disclosure, use, exhibition or other means
- *P* document published prior to the international filing date but later than the priority date claimed

- *T* later document published after the international filing date or priority date and not in conflict with the application but cited to understand the principle or theory underlying the invention
- *X* document of particular relevance; the claimed invention cannot be considered novel or cannot be considered to involve an inventive step when the document is taken alone
- *Y* document of particular relevance; the claimed invention cannot be considered to involve an inventive step when the document is combined with one or more other such documents, such combination being obvious to a person skilled in the art.
- *Z* document member of the same patent family

Date of the actual completion of the international search

29 August 2000

Date of mailing of the international search report

31.10.2000

Name and mailing address of the ISA

European Patent Office, P.B. 5818 Patentlaan 2
NL - 2280 HV Rijswijk
Tel. (+31-70) 340-2040, Tx. 31 651 epo nl,
Fax: (+31-70) 340-3016

Authorized officer

Mans Marklund

INTERNATIONAL SEARCH REPORT

International Application No
PCT/NO 00/00144

C.(Continuation) DOCUMENTS CONSIDERED TO BE RELEVANT

Category *	Citation of document, with indication, where appropriate, of the relevant passages	Relevant to claim No.
P,A	EP 0 989 705 A (AT & T CORP) 29 March 2000 (2000-03-29) the whole document -----	1

INTERNATIONAL SEARCH REPORT

Information on patent family members

International Application No
PCT/NO 00/00144

Patent document cited in search report		Publication date	Patent family member(s)	Publication date
EP 1003343	A	24-05-2000	NONE	
EP 0996270	A	26-04-2000	NONE	
EP 0989705	A	29-03-2000	CN 1250996 A	19-04-2000



INTERNATIONAL APPLICATION PUBLISHED UNDER THE PATENT COOPERATION TREATY (PCT)

(51) International Patent Classification ⁶ : G06F 15/16	A1	(11) International Publication Number: WO 00/28431 (43) International Publication Date: 18 May 2000 (18.05.00)
--	-----------	--

(21) International Application Number: PCT/US99/24561

(22) International Filing Date: 21 October 1999 (21.10.99)

(30) Priority Data:
60/107,167 5 November 1998 (05.11.98) US
09/405,318 23 September 1999 (23.09.99) US

(71) Applicant: BEA SYSTEMS, INC. [US/US]; 2315 North First Street, San Jose, CA 95131 (US).

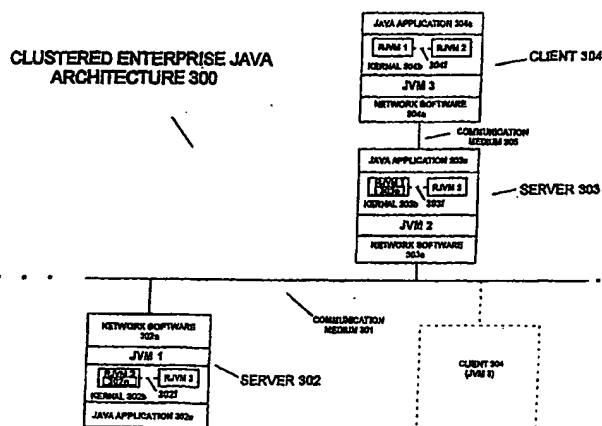
(72) Inventors: JACOBS, Dean, B.; 1747 Madera Street, Berkeley, CA 94707 (US). LANGEN, Anno, R.; 2220-d Sacramento Street, Berkeley, CA 94702 (US).

(74) Agents: MEYER, Sheldon, R. et al.; Fliesler Dubb Meyer & Lovejoy LLP, Suite 400, Four Embarcadero Center, San Francisco, CA 94111-4156 (US).

(81) Designated States: AE, AL, AM, AT, AU, AZ, BA, BB, BG, BR, BY, CA, CH, CN, CR, CU, CZ, DE, DK, EE, ES, FI, GB, GD, GE, GH, GM, HR, HU, ID, IL, IN, IS, JP, KE, KG, KP, KR, KZ, LC, LK, LR, LS, LT, LU, LV, MD, MG, MK, MN, MW, MX, NO, NZ, PL, PT, RO, RU, SD, SE, SG, SI, SK, SL, TJ, TM, TR, TT, UA, UG, UZ, VN, YU, ZA, ZW, ARIPO patent (GH, GM, KE, LS, MW, SD, SL, SZ, TZ, UG, ZW), Eurasian patent (AM, AZ, BY, KG, KZ, MD, RU, TJ, TM), European patent (AT, BE, CH, CY, DE, DK, ES, FI, FR, GB, GR, IE, IT, LU, MC, NL, PT, SE), OAPI patent (BF, BJ, CF, CG, CI, CM, GA, GN, GW, ML, MR, NE, SN, TD, TG).

Published*With international search report.**Before the expiration of the time limit for amending the claims and to be republished in the event of the receipt of amendments.*

(54) Title: CLUSTERED ENTERPRISE JAVA™ HAVING A MESSAGE PASSING KERNEL IN A DISTRIBUTED PROCESSING SYSTEM

**(57) Abstract**

A clustered enterprise Java™ distributed processing system is provided. The distributed processing system includes a first and a second computer coupled to a communication medium. The first computer includes a Java™ virtual machine (JVM) and kernel software layer for transferring messages, including a remote Java™ virtual machine (RJVM). The second computer includes a JVM and a kernel software layer having a RJVM. Messages are passed from a RJVM to the JVM in one computer to the JVM and RJVM in the second computer. Messages may be forwarded through an intermediate server or rerouted after a network reconfiguration. Each computer includes a Smart stub having a replica handler, including a load balancing software component and a failover software component. Each computer includes a duplicated service naming tree for storing a pool of Smart stubs at a node. The computers may be programmed in a stateless, stateless factory, or a stateful programming model. The clustered enterprise Java™ distributed processing system allows for enhanced scalability and fault tolerance.

FOR THE PURPOSES OF INFORMATION ONLY

Codes used to identify States party to the PCT on the front pages of pamphlets publishing international applications under the PCT.

AL	Albania	ES	Spain	LS	Lesotho	SI	Slovenia
AM	Armenia	FI	Finland	LT	Lithuania	SK	Slovakia
AT	Austria	FR	France	LU	Luxembourg	SN	Senegal
AU	Australia	GA	Gabon	LV	Latvia	SZ	Swaziland
AZ	Azerbaijan	GB	United Kingdom	MC	Monaco	TD	Chad
BA	Bosnia and Herzegovina	GE	Georgia	MD	Republic of Moldova	TG	Togo
BB	Barbados	GH	Ghana	MG	Madagascar	TJ	Tajikistan
BE	Belgium	GN	Guinea	MK	The former Yugoslav Republic of Macedonia	TM	Turkmenistan
BF	Burkina Faso	GR	Greece	ML	Mali	TR	Turkey
BG	Bulgaria	HU	Hungary	MN	Mongolia	TT	Trinidad and Tobago
BJ	Benin	IE	Ireland	MR	Mauritania	UA	Ukraine
BR	Brazil	IL	Israel	MW	Malawi	UG	Uganda
BY	Belarus	IS	Iceland	MX	Mexico	US	United States of America
CA	Canada	IT	Italy	NE	Niger	UZ	Uzbekistan
CF	Central African Republic	JP	Japan	NL	Netherlands	VN	Viet Nam
CG	Congo	KE	Kenya	NO	Norway	YU	Yugoslavia
CH	Switzerland	KG	Kyrgyzstan	NZ	New Zealand	ZW	Zimbabwe
CI	Côte d'Ivoire	KP	Democratic People's Republic of Korea	PL	Poland		
CM	Cameroon	KR	Republic of Korea	PT	Portugal		
CN	China	KZ	Kazakhstan	RO	Romania		
CU	Cuba	LC	Saint Lucia	RU	Russian Federation		
CZ	Czech Republic	LI	Liechtenstein	SD	Sudan		
DE	Germany	LK	Sri Lanka	SE	Sweden		
DK	Denmark	LR	Liberia	SG	Singapore		
EE	Estonia						

**CLUSTERED ENTERPRISE JAVA™ HAVING A MESSAGE PASSING
KERNEL IN A DISTRIBUTED PROCESSING SYSTEM**

Field of the Invention

5 The present invention relates to distributed processing systems
and, in particular, computer software in distributed processing
systems.

Cross Reference to Related Applications

 This application claims the benefit of U.S. Provisional
Application No. 60/107,167, filed November 5, 1998.

10 The following copending U.S. patent applications are assigned
to the assignee of the present application, and their disclosures are
incorporated herein by reference:

 (A) Ser. No. Not Yet Known [Attorney Docket No. BEAS1029]
filed Not Yet Known by Dean B. Jacobs and Eric M. Halpern, and
15 entitled, "A SMART STUB OR ENTERPRISE JAVA™ BEAN IN A
DISTRIBUTED PROCESSING SYSTEM";

 (B) Ser. No. Not Yet Known [Attorney Docket No. BEAS1030]
filed Not Yet Known by Dean B. Jacobs and Eric M. Halpern, and
entitled, "A DUPLICATED NAMING SERVICE IN A DISTRIBUTED
20 PROCESSING SYSTEM"; and

 (C) Ser. No. Not Yet Known [Attorney Docket No. BEAS1031]
filed Not Yet Known by Dean B. Jacobs and Anno R. Langer, and
originally entitled, "CLUSTERED ENTERPRISE JAVA™ IN A SECURE
DISTRIBUTED PROCESSING SYSTEM".

25 **Background of the Invention**

 There are several types of distributed processing systems.
Generally, a distributed processing system includes a plurality of

processing devices, such as two computers coupled to a communication medium. Communication mediums may include wired mediums, wireless mediums, or combinations thereof, such as an Ethernet local area network or a cellular network. In a distributed processing system, at least one processing device may transfer information on the communication medium to another processing device.

Client/server architecture 110 illustrated in Fig. 1a is one type of distributed processing system. Client/server architecture 110 includes at least two processing devices, illustrated as client 105 and application server 103. Additional clients may also be coupled to communication medium 104, such as client 108.

Typically, server 103 hosts business logic and/or coordinates transactions in providing a service to another processing device, such as client 103 and/or client 108. Application server 103 is typically programmed with software for providing a service. The software may be programmed using a variety of programming models, such as Enterprise Java™ Bean ("EJB") 100b as illustrated in Figs. 1a-b. The service may include, for example, retrieving and transferring data from a database, providing an image and/or calculating an equation. For example, server 103 may retrieve data from database 101a in persistent storage 101 over communication medium 102 in response to a request from client 105. Application server 103 then may transfer the requested data over communication medium 104 to client 105.

A client is a processing device which utilizes a service from a server and may request the service. Often a user 106 interacts with client 106 and may cause client 105 to request service over a communication medium 104 from application server 103. A client

often handles direct interactions with end users, such as accepting requests and displaying results.

A variety of different types of software may be used to program application server 103 and/or client 105. One programming language is the Java™ programming language. Java™ application object code is loaded into a Java™ virtual machine ("JVM"). A JVM is a program loaded onto a processing device which emulates a particular machine or processing device. More information on the Java™ programming language may be obtained at <http://www.javasoft.com>, which is incorporated by reference herein.

Fig. 1b illustrates several Java™ Enterprise Application Programming Interfaces ("APIs") 100 that allow Java™ application code to remain independent from underlying transaction systems, data-bases and network infrastructure. Java™ Enterprise APIs 100 include, for example, remote method invocation ("RMI") 100a, EJBs 100b, and Java™ Naming and Directory Interface (JNDI) 100c.

RMI 100a is a distributed programming model often used in peer-to-peer architecture described below. In particular, a set of classes and interfaces enables one Java™ object to call the public method of another Java™ object running on a different JVM.

An instance of EJB 100b is typically used in a client/server architecture described above. An instance of EJB 100b is a software component or a reusable pre-built piece of encapsulated application code that can be combined with other components. Typically, an instance of EJB 100b contains business logic. An EJB 100b instance stored on server 103 typically manages persistence, transactions, concurrency, threading, and security.

JNDI 100c provides directory and naming functions to Java™ software applications.

Client/server architecture 110 has many disadvantages. First, architecture 110 does not scale well because server 103 has to handle many connections. In other words, the number of clients which may be added to server 103 is limited. In addition, adding
5 twice as many processing devices (clients) does not necessarily provide you with twice as much performance. Second, it is difficult to maintain application code on clients 105 and 108. Third, architecture 110 is susceptible to system failures or a single point of failure. If server 101 fails and a backup is not available, client 105
10 will not be able to obtain the service.

Fig. 1c illustrates a multi-tier architecture 160. Clients 151, 152 manage direct interactions with end users, accepting requests and display results. Application server 153 hosts the application code, coordinates communications, synchronizations, and
15 transactions. Database server 154 and portable storage device 155 provides durable transactional management of the data.

Multi-tier architecture 160 has similar client/server architecture 110 disadvantages described above.

Fig. 2 illustrates peer-to-peer architecture 214. Processing
20 devices 216, 217 and 218 are coupled to communication medium 213. Processing devices 216, 217, and 218 include network software 210a, 210b, and 210c for communicating over medium 213. Typically, each processing device in a peer-to-peer architecture has similar processing capabilities and applications. Examples of
25 peer-to-peer program models include Common Object Request Broker Architecture ("CORBA") and Distributed Object Component Model ("DCOM") architecture.

In a platform specific distributed processing system, each processing device may run the same operating system. This allows

the use of proprietary hardware, such as shared disks, multi-tailed disks, and high speed interconnects, for communicating between processing devices. Examples of platform-specific distributed processing systems include IBM® Corporation's S/390® Parallel Sysplex®, Compaq's Tandem Division Himalaya servers, Compaq's Digital Equipment Corporation™ (DEC™) Division OpenVMS™ Cluster software, and Microsoft® Corporation Windows NT® cluster services (Wolfpack).

Fig. 2b illustrates a transaction processing (TP) architecture 220. In particular, TP architecture 220 illustrates a BEA® Systems, Inc. TUXEDO® architecture. TP monitor 224 is coupled to processing devices ATM 221, PC 222, and TP monitor 223 by communication medium 280, 281, and 282, respectively. ATM 221 may be an automated teller machine, PC 222 may be a personal computer, and TP monitor 223 may be another transaction processor monitor. TP monitor 224 is coupled to back-end servers 225, 226, and 227 by communication mediums 283, 284, and 285. Server 225 is coupled to persistent storage device 287, storing database 289, by communication medium 286. TP monitor 224 includes a workflow controller 224a for routing service requests from processing devices, such as ATM 221, PC 222, or TP monitor 223, to various servers such as server 225, 226 and 227. Work flow controller 224a enables (1) workload balancing between servers, (2) limited scalability or allowing for additional servers and/or clients, (3) fault tolerance of redundant backend servers (or a service request may be sent by a workflow controller to a server which has not failed), and (4) session concentration to limit the number of simultaneous connections to back-end servers. Examples of other transaction processing architectures include IBM® Corporation's CICS® ,

Compaq's Tandem Division Pathway/Ford/TS, Compaq's DEC™ ACMS, and Transarc Corporation's Encina.

TP architecture 220 also has many disadvantages. First, a failure of a single processing device or TP monitor 224 may render the network inoperable. Second, the scalability or number of processing devices (both servers and clients) coupled to TP monitor 224 may be limited by TP monitor 224 hardware and software. Third, flexibility in routing a client request to a server is limited. For example, if communication medium 280 is inoperable, but communication medium 290 becomes available, ATM 221 typically may not request service directly from server 225 over communication medium 290 and must access TP monitor 224. Fourth, a client typically does not know the state of a back-end server or other processing device. Fifth, no industry standard software or APIs are used for load balancing. And sixth, a client typically may not select a particular server even if the client has relevant information which would enable efficient service.

Therefore, it is desirable to provide a distributed processing system and, in particular, distributed processing system software that has the advantages of the prior art distributed processing systems without the inherent disadvantages. The software should allow for industry standard APIs which are typically used in either client/server, multi-tier, or peer-to-peer distributed processing systems. The software should support a variety of computer programming models. Further, the software should enable (1) enhanced fault tolerance, (2) efficient scalability, (3) effective load balancing, and (4) session concentration control. The improved computer software should allow for rerouting or network reconfiguration. Also, the computer

software should allow for the determination of the state of a processing device.

SUMMARY OF THE INVENTION

5 An improved distributed processing system is provided and, in particular, computer software for a distributed processing system is provided. The computer software improves the fault tolerance of the distributed processing system as well as enables efficient scalability. The computer software allows for efficient load balancing and session concentration. The computer software supports rerouting or reconfiguration of a computer network. The computer software supports a variety of computer programming models and allows for the use of industry standard APIs that are used in both client/server and peer-to-peer distributed processing architectures. The computer software enables a determination of the state of a server or other processing device. The computer software also supports message forwarding under a variety of circumstances, including a security model.

20 According to one aspect of the present invention, a distributed processing system comprises a communication medium coupled to a first processing device and a second processing device. The first processing device includes a first software program emulating a processing device ("JVM1") including a first kernel software layer having a data structure ("RJVM1"). The second processing device includes a first software program emulating a processing device ("JVM2") including a first kernel software layer having a data structure ("RJVM2"). A message from the first processing device is transferred to the second processing device through the first kernel software layer and the first software program in the first processing

device to the first kernel software layer and the first software program in the second processing device.

According to another aspect of the present invention, the first software program in the first processing device is a Java™ virtual machine ("JVM") and the data structure in the first processing device is a remote Java™ virtual machine ("RJVM"). Similarly, the first software program in the second processing device is a JVM and the data structure in the second processing device is a RJVM. The RJVM in the second processing device corresponds to the JVM in the first processing device.

According to another aspect of the present invention, the RJVM in the first processing device includes a socket manager software component, a thread manager software component, a message routing software component, a message compression software component, and/or a peer-gone detection software component.

According to another aspect of the present invention, the first processing device communicates with the second processing device using a protocol selected from the group consisting of Transmission Control Protocol ("TCP"), Secure Sockets Layer ("SSL"), Hypertext Transport Protocol ("HTTP") tunneling, and Internet InterORB Protocol ("IIOP") tunneling.

According to another aspect of the present invention, the first processing device includes memory storage for a Java™ application.

According to another aspect of the present invention, the first processing device is a peer of the second processing device. Also, the first processing device is a server and the second processing device is a client.

According to another aspect of the present invention, a second communication medium is coupled to the second processing device. A third processing device is coupled to the second communication medium. The third processing device includes a first software
5 program emulating a processing device ("JVM3"), including a kernel software layer having a first data structure ("RJVM1"), and a second data structure ("RJVM2").

According to still another aspect of the present invention, the first processing device includes a stub having a replica-handler
10 software component. The replica-handler software component includes a load balancing software component and a failover software component.

According to another aspect of the present invention, the first processing device includes an Enterprise Java™ Bean object.

15 According to still another aspect of the present invention, the first processing device includes a naming tree having a pool of stubs stored at a node of the tree and the second processing device includes a duplicate of the naming tree.

According to still another aspect of the present invention, the
20 first processing device includes an application program coded in a stateless program model and the application program includes a stateless session bean.

According to still another aspect of the present invention, the first processing device includes an application program coded in a
25 stateless factory program model and the application program includes a stateful session bean.

According to still another aspect of the present invention, the first processing device includes an application program coded in a

stateful program model and the application program includes an entity session bean.

According to still another aspect of the present invention, an article of manufacture including an information storage medium is provided. The article of manufacture comprises a first set of digital information for transferring a message from a RJVM in a first processing device to a RJVM in a second processing device.

According to another aspect of the present invention, the article of manufacture comprises a first set of digital information, including a stub having a load balancing software program for selecting a service provider from a plurality of service providers.

According to another aspect of the present invention, the stub has a failover software component for removing a failed service provider from the plurality of service providers.

According to another aspect of the present invention, the load balancing software component selects a service provider based on an affinity for a particular service provider.

According to another aspect of the present invention, the load balancing software component selects a service provider in a round robin manner.

According to another aspect of the present invention, the load balancing software component randomly selects a service provider.

According to another aspect of the present invention, the load balancing software component selects a service provider from the plurality of service providers based upon the load of each service provider.

According to another aspect of the present invention, the load balancing software component selects a service provider from the plurality of service providers based upon the data type requested.

According to another aspect of the present invention, the load balancing software component selects a service provider from the plurality of service providers based upon the closest physical service provider.

5 According to another aspect of the present invention, the load balancing software component selects a service provider from the plurality of service providers based upon a time period in which each service provider responds.

10 According to another aspect of the present invention, the article of manufacture comprises a first set of digital information, including an Enterprise Java™ Bean object for selecting a service provider from a plurality of service providers.

15 According to another aspect of the present invention, a stub is stored in a processing device in a distributed processing system. The stub includes a method comprising the steps of obtaining a list of service providers and selecting a service provider from the list of service providers.

20 According to another aspect of the present invention, the method further includes removing a failed service provider from the list of service providers.

25 According to still another aspect of the present invention, an apparatus comprises a communication medium coupled to a first processing device and a second processing device. The first processing device stores a naming tree including a remote method invocation ("RMI") stub for accessing a service provider. The second processing device has a duplicate naming tree and the service provider.

According to another aspect of the present invention, the naming tree has a node including a service pool of current service providers.

5 According to another aspect of the present invention, the service pool includes a stub.

According to another aspect of the present invention, a distributed processing system comprises a first computer coupled to a second computer. The first computer has a naming tree, including a remote invocation stub for accessing a service provider. The
10 second computer has a replicated naming tree and the service provider.

According to another aspect of the present invention, a distributed processing system comprising a first processing device coupled to a second processing device is provided. The first
15 processing device has a JVM and a first kernel software layer including a first RJVM. The second processing device includes a first JVM and a first kernel software layer including a second RJVM. A message may be transferred from the first processing device to the second processing device when there is not a socket available
20 between the first JVM and the second JVM.

According to another aspect of the present invention, the first processing device is running under an applet security model, behind a firewall or is a client, and the second processing device is also a client.

25 Other aspects and advantages of the present invention can be seen upon review of the figures, the detailed description, and the claims which follow.

Brief Description of the Figures

Fig. 1a illustrates a prior art client/server architecture;

Fig. 1b illustrates a prior art Java™ enterprise APIs;

Fig. 1c illustrates a multi-tier architecture;

5 Fig. 2a illustrates a prior art peer-to-peer architecture;

Fig. 2b illustrates a prior art transaction processing architecture;

Fig. 3a illustrates a simplified software block diagram of an embodiment of the present invention;

10 Fig. 3b illustrates a simplified software block diagram of the kernel illustrated in Fig. 3a;

Fig. 3c illustrates a clustered enterprise Java™ architecture;

Fig. 4 illustrates a clustered enterprise Java™ naming service architecture;

15 Fig. 5a illustrates a smart stub architecture;

Fig. 5b illustrates an EJB object architecture;

Fig. 6a is a control flow chart illustrating a load balancing method;

20 Figs. 6b-g are control flow charts illustrating load balancing methods;

Fig. 7 is a control flow chart illustrating a failover method;

Fig. 8 illustrates hardware and software components of a client/server in the clustered enterprise Java™ architecture shown in Figs. 3-5.

25 The invention will be better understood with reference to the drawings and detailed description below. In the drawings, like reference numerals indicate like components.

DETAILED DESCRIPTION**I. Clustered Enterprise Java™ Distributed Processing System****A. Clustered Enterprise Java™ Software Architecture**

Fig. 3a illustrates a simplified block diagram 380 of the software layers in a processing device of a clustered enterprise Java™ system, according to an embodiment of the present invention. A detailed description of a clustered enterprise Java™ distributed processing system is described below. The first layer of software includes a communication medium software driver 351 for transferring and receiving information on a communication medium, such as an ethernet local area network. An operating system 310 including a transmission control protocol ("TCP") software component 353 and internet protocol ("IP") software component 352 are upper software layers for retrieving and sending packages or blocks of information in a particular format. An "upper" software layer is generally defined as a software component which utilizes or accesses one or more "lower" software layers or software components. A JVM 354 is then implemented. A kernel 355 having a remote Java™ virtual machine 356 is then layered on JVM 354. Kernel 355, described in detail below, is used to transfer messages between processing devices in a clustered enterprise Java™ distributed processing system. Remote method invocation 357 and enterprise Java™ bean 358 are upper software layers of kernel 355. EJB 358 is a container for a variety of Java™ applications.

Fig. 3b illustrates a detailed view of kernel 355 illustrated in Fig. 3a. Kernel 355 includes a socket manager component 363, thread manager 364 component, and RJVM 356. RJVM 356 is a data structure including message routing software component 360, message compression software component 361 including

abbreviation table 161c, and peer-gone detection software component 362. RJVM 356 and thread manager component 364 interact with socket manager component 363 to transfer information between processing devices.

5 **B. Distributed Processing System**

Fig. 3 illustrates a simplified block diagram of a clustered enterprise Java™ distributed processing system 300. Processing devices are coupled to communication medium 301. Communication medium 301 may be a wired and/or wireless communication medium or combination thereof. In an embodiment, communication medium 10 301 is a local-area-network (LAN). In an alternate embodiment, communication medium 301 is a world-area-network (WAN) such as the Internet or World Wide Web. In still another embodiment, communication medium 301 is both a LAN and a WAN.

15 A variety of different types of processing devices may be coupled to communication medium 301. In an embodiment, a processing device may be a general purpose computer 100 as illustrated in Fig. 8 and described below. One of ordinary skill in the art would understand that Fig. 8 and the below description describes 20 one particular type of processing device where multiple other types of processing devices with a different software and hardware configurations could be utilized in accordance with an embodiment of the present invention. In an alternate embodiment, a processing device may be a printer, handheld computer, laptop computer, 25 scanner, cellular telephone, pager, or equivalent thereof.

Fig. 3c illustrates an embodiment of the present invention in which servers 302 and 303 are coupled to communication medium 301. Server 303 is also coupled to communication medium 305

which may have similar embodiments as described above in regard to communication medium 301. Client 304 is also coupled to communication medium 305. In an alternate embodiment, client 304 may be coupled to communication medium 301 as illustrated by the dashed line and box in Fig. 3c. It should be understood that in alternate embodiments, server 302 is (1) both a client and a server, or (2) a client. Similarly, Fig. 3 illustrates an embodiment in which three processing devices are shown wherein other embodiments of the present invention include multiple other processing devices or communication mediums as illustrated by the ellipses.

Server 302 transfers information over communication medium 301 to server 303 by using network software 302a and network software 303a, respectively. In an embodiment, network software 302a, 303a, and 304a include communication medium software driver 351, Transmission Control Protocol software 353, and Internet Protocol software 352 ("TCP/IP"). Client 304 also includes network software 304a for transferring information to server 303 over communication medium 305. Network software 303a in server 303 is also used to transfer information to client 304 by way of communication medium 305.

According to an embodiment of the present invention, each processing device in clustered enterprise Java™ architecture 300 includes a message-passing kernel 355 that supports both multi-tier and peer-to-peer functionality. A kernel is a software program used to provide fundamental services to other software programs on a processing device.

In particular, server 302, server 303, and client 304 have kernels 302b, 303b, and 304b, respectively. In particular, in order for two JVMs to interact, whether they are clients or servers, each

JVM constructs an RJVM representing the other. Messages are sent from the upper layer on one side, through a corresponding RJVM, across the communication medium, through the peer RJVM, and delivered to the upper layer on the other side. In various
5 embodiments, messages can be transferred using a variety of different protocols, including, but not limited to, Transmission Control Protocol/Internet Protocol ("TCP/IP"), Secure Sockets Layer ("SSL"), Hypertext Transport Protocol ("HTTP") tunneling, and Internet
10 InterORB Protocol ("IIOP") tunneling, and combinations thereof. The RJVMs and socket managers create and maintain the sockets underlying these protocols and share them between all objects in the upper layers. A socket is a logical location representing a terminal between processing devices in a distributed processing system. The
15 kernel maintains a pool of execute threads and thread manager software component 364 multiplexes the threads between socket reading and request execution. A thread is a sequence of executing program code segments or functions.

For example, server 302 includes JVM1 and Java™ application 302c. Server 302 also includes a RJVM2 representing the JVM2 of
20 server 303. If a message is to be sent from server 302 to server 303, the message is sent through RJVM2 in server 302 to RJVM1 in server 303.

C. Message Forwarding

Clustered enterprise Java™ network 300 is able to forward a
25 message through an intermediate server. This functionality is important if a client requests a service from a back-end server through a front-end gateway. For example, a message from server 302 (client 302) and, in particular, JVM1 may be forwarded to client

304 (back-end server 304) or JVM3 through server 303 (front-end gateway) or JVM2. This functionality is important in controlling session concentration or how many connections are established between a server and various clients.

5 Further, message forwarding may be used in circumstances where a socket cannot be created between two JVMs. For example, a sender of a message is running under the applet security model which does not allow for a socket to be created to the original server. A detailed description of the applet security model is provided at
10 <http://www.javasoft.com>, which is incorporated herein by reference. Another example includes when the receiver of the message is behind a firewall. Also, as described below, message forwarding is applicable if the sender is a client and the receiver is a client and thus does not accept incoming sockets.

15 For example, if a message is sent from server 302 to client 304, the message would have to be routed through server 303. In particular, a message handoff, as illustrated by 302f, between RJVM3 (representing client 304) would be made to RJVM2 (representing server 303) in server 302. The message would be
20 transferred using sockets 302e between RJVM2 in server 302 and RJVM1 in server 303. The message would then be handed off, as illustrated by dashed line 303f, from RJVM1 to RJVM3 in server 303. The message would then be passed between sockets of RJVM3 in server 303 and RJVM2 in client 304. The message then would be
25 passed, as illustrated by the dashed line 304f, from RJVM2 in client 304 to RJVM1 in client 304.

D. Rerouting

An RJVM in client/server is able to switch communication paths or communication mediums to other RJVMs at any time. For example, if client 304 creates a direct socket to server 302, server
5 302 is able to start using the socket instead of message forwarding through server 303. This embodiment is illustrated by a dashed line and box representing client 304. In an embodiment, the use of transferring messages by RJVMs ensures reliable, in-order message delivery after the occurrence of a network reconfiguration. For
10 example, if client 304 was reconfigured to communication medium 301 instead of communication medium 305 as illustrated in Fig. 3. In an alternate embodiment, messages may not be delivered in order.

An RJVM performs several end-to-end operations that are carried through routing. First, an RJVM is responsible for detecting
15 when a respective client/server has unexpectedly died. In an embodiment, peer-gone selection software component 362, as illustrated in Fig. 3b, is responsible for this function. In an embodiment, an RJVM sends a heartbeat message to other clients/servers when no other message has been sent in a
20 predetermined time period. If the client/server does not receive a heartbeat message in the predetermined count time, a failed client/server which should have sent the heartbeat, is detected. In an embodiment, a failed client/server is detected by connection timeouts or if no messages have been sent by the failed client/server
25 in a predetermined amount of time. In still another embodiment, a failed socket indicates a failed server/client.

Second, during message serialization, RJVMs, in particular, message compression software 360, abbreviate commonly transmitted data values to reduce message size. To accomplish this,

each JVM/RJVM pair maintains matching abbreviation tables. For example, JVM1 includes an abbreviation table and RJVM1 includes a matching abbreviation table. During message forwarding between an intermediate server, the body of a message is not deserialized on the intermediate server in route.

E. Multi-tier/Peer-to-Peer Functionality

Clustered enterprise Java™ architecture 300 allows for multi-tier and peer-to-peer programming.

Clustered enterprise Java™ architecture 300 supports an explicit syntax for client/server programming consistent with a multi-tier distributed processing architecture. As an example, the following client-side code fragment writes an informational message to a server's log file:

```
T3Client clnt = new T3Client("t3://acme:7001");
LogServices log = clnt.getT3Services().log();
log.info("Hello from a client");
```

The first line establishes a session with the acme server using the t3 protocol. If RJVMs do not already exist, each JVM constructs an RJVM for the other and an underlying TCP socket is established. The client-side representation of this session - the T3Client object - and the server-side representation communicate through these RJVMs. The server-side supports a variety of services, including database access, remote file access, workspaces, events, and logging. The second line obtains a LogServices object and the third line writes the message.

Clustered enterprise Java™ computer architecture 300 also supports a server-neutral syntax consistent with a peer-to-peer distributed processing architecture. As an example, the following

code fragment obtains a stub for an RMI object from the JNDI-compliant naming service on a server and invokes one of its methods.

```
        Hashtable env = new Hashtable();  
        env.put(Context.PROVIDER_URL, "t3://acme:7001");  
5      env.put(Context.INITIAL_CONTEXT_FACTORY,  
        "weblogic.jndi.WebLogicInitialContextFactory");  
        Context ctx = new InitialContext(env);  
        Example e = (Example) ctx.lookup("acme.eng.example");  
        result = e.example(37);
```

10 In an embodiment, JNDI naming contexts are packaged as RMI objects to implement remote access. Thus, the above code illustrates a kind of RMI bootstrapping. The first four lines obtain an RMI stub for the initial context on the acme server. If RJVMs do not already exist, each side constructs an RJVM for the other and an
15 underlying TCP socket for the t3 protocol is established. The caller-side object - the RMI stub - and the callee-side object - an RMI impl - communicate through the RJVMs. The fifth line looks up another RMI object, an Example, at the name acme.eng.example and the sixth
20 line invokes one of the Example methods. In an embodiment, the Example impl is not on the same processing device as the naming service. In another embodiment, the Example impl is on a client. Invocation of the Example object leads to the creation of the appropriate RJVMs if they do not already exist.

II. Replica-Aware or Smart Stubs/EJB Objects

25 In Fig. 3c, a processing device is able to provide a service to other processing devices in architecture 300 by replicating RMI and/or EJB objects. Thus, architecture 300 is easily scalable and

fault tolerant. An additional service may easily be added to architecture 300 by adding replicated RMI and/or EJB objects to an existing processing device or newly added processing device.

Moreover, because the RMI and/or EJB objects can be replicated
5 throughout architecture 300, a single processing device, multiple processing devices, and/or a communication medium may fail and still not render architecture 300 inoperable or significantly degraded.

Fig. 5a illustrates a replica-aware ("RA") or Smart stub 580 in architecture 500. Architecture 500 includes client 504 coupled to
10 communication medium 501. Servers 502 and 503 are coupled to communication medium 501, respectively. Persistent storage device 509 is coupled to server 502 and 503 by communication medium 560 and 561, respectively. In various embodiments, communication medium 501, 560, and 561 may be wired and/or wireless
15 communication mediums as described above. Similarly, in an embodiment, client 504, server 502, and server 503 may be both clients and servers as described above. One of ordinary skill in the art would understand that in alternate embodiments, multiple other servers and clients may be included in architecture 500 as illustrated
20 by ellipses. Also, as stated above, in alternate embodiments, the hardware and software configuration of client 504, server 502 and server 503 is described below and illustrated in Fig. 8.

RA RMI stub 580 is a Smart stub which is able to find out about all of the service providers and switch between them based on
25 a load balancing method 507 and/or failover method 508. In an embodiment, an RA stub 580 includes a replica handler 506 that selects an appropriate load balancing method 507 and/or failover method 507. In an alternate embodiment, a single load balancing method and/or single failover method is implemented. In alternate

embodiments, replica handler 506 may include multiple load balancing methods and/or multiple failover methods and combinations thereof. In an embodiment, a replica handler 506 implements the following interface:

```
5      public interface ReplicaHandler {  
          Object loadBalance(Object currentProvider) throws  
          RefreshAbortedException;  
          Object failOver(Object failedProvider,  
10         RemoteException e) throws  
          RemoteException;  
      }
```

Immediately before invoking a method, RA stub 580 calls load balance method 507, which takes the current server and returns a replacement. For example, client 504 may be using server 502 for
15 retrieving data for database 509a or personal storage device 509. Load balance method 507 may switch to server 503 because server 502 is overloaded with service requests. Handler 506 may choose a server replacement entirely on the caller, perhaps using information about server 502 load, or handler 506 may request server 502 for
20 retrieving a particular type of data. For example, handler 506 may select a particular server for calculating an equation because the server has enhanced calculation capability. In an embodiment, replica handler 506 need not actually switch providers on every invocation because replica handler 506 is trying to minimize the
25 number of connections that are created.

Fig. 6a is a control flow diagram illustrating the load balancing software 507 illustrated in Figs. 5a-b. It should be understood that Fig. 6a is a control flow diagram illustrating the logical sequence of

functions or steps which are completed by software in load balancing method 507. In alternate embodiments, additional functions or steps are completed. Further, in an alternate embodiment, hardware may perform a particular function or all the functions.

5 Load balancing software 507 begins as indicated by circle 600. A determination is then made in logic block 601 as to whether the calling thread established "an affinity" for a particular server. A client has an affinity for the server that coordinates its current transaction and a server has an affinity for itself. If an affinity is
10 established, control is passed to logic block 602, otherwise control is passed to logic block 604. A determination is made in logic block 602 whether the affinity server provides the service requested. If so, control is passed to logic block 603. Otherwise, control is passed to logic block 604. The provider of the service on the affinity server is
15 returned to the client in logic block 603. In logic block 604, a naming service is contacted and an updated list of the current service providers is obtained. A getNextProvider method is called to obtain a service provider in logic block 605. Various embodiments of the getNextProvider method are illustrated in Figs. 6b-g and described in
20 detail below. The service is obtained in logic block 606. Failover method 508 is then called if service is not provided in logic block 606 and load balancing method 507 exits as illustrated by logic block 608. An embodiment of failover method 508 is illustrated in Fig. 7 and described in detail below.

25 Figs. 6b-g illustrate various embodiments of a getNextProvider method used in logic block 605 of Fig. 6a. As illustrated in Fig. 6b, the getNextProvider method selects a service provider in a round robin manner. A getNextProvider method 620 is entered as illustrated by circle 621. A list of current service providers is

obtained in logic block 622. A pointer is incremented in logic block 623. The next service provider is selected based upon the pointer in logic block 624 and the new service provider is returned in logic block 625 and getNextProvider method 620 exits as illustrated by
5 circle 626.

Fig. 6c illustrates an alternate embodiment of a getNextProvider method which obtains a service provider by selecting a service provider randomly. A getNextProvider method 630 is entered as illustrated by circle 631. A list of current service
10 providers is obtained as illustrated by logic block 632. The next service provider is selected randomly as illustrated by logic block 633 and a new service provider is returned in logic block 634. The getNextProvider method 630 then exits, as illustrated by circle 635.

Still another embodiment of a getNextProvider method is
15 illustrated in Fig. 6d which obtains a service provider based upon the load of the service providers. A getNextProvider method 640 is entered as illustrated by circle 641. A list of current service providers is obtained in logic block 642. The load of each service provider is obtained in logic block 643. The service provider with the
20 least load is then selected in logic block 644. The new service provider is then returned in logic block 645 and getNextProvider method 640 exits as illustrated by circle 646.

An alternate embodiment of a getNextProvider method is illustrated in Fig. 6e which obtains a service provider based upon the
25 type of data obtained from the service provider. A getNextProvider method 650 is entered as illustrated by circle 651. A list of current service providers is obtained in logic block 652. The type of data requested from the service providers is determined in logic block 653. The service provider is then selected based on the data type in logic

block 654. The service provider is returned in logic block 655 and getNextProvider method 650 exits as illustrated by circle 656.

Still another embodiment of a getNextProvider method is illustrated in Fig. 6f which selects a service provider based upon the physical location of the service providers. A getNextProvider method 5 660 is entered as illustrated by circle 661. A list of service providers is obtained as illustrated by logic block 662. The physical distance to each service provider is determined in logic block 663 and the service provider which has the closest physical distance to the requesting 10 client is selected in logic block 664. The new service provider is then returned in logic block 665 and the getNextProvider method 660 exits as illustrated by circle 666.

Still a further embodiment of the getNextProvider method is illustrated in Fig. 6g and selects a service provider based on the amount of time taken for the service provider to respond to previous 15 requests. Control of getNextProvider method 670 is entered as illustrated by circle 671. A list of current service providers is obtained in logic block 672. The time period for each service provider to respond to a particular message is determined in logic 20 block 673. The service provider which responds in the shortest time period is selected in logic block 674. The new service provider is then returned in logic block 675 and control from getNextProvider method 670 exits as illustrated by circle 676.

If invocation of a service method fails in such a way that a 25 retry is warranted, RA 580 stub calls failover method 508, which takes the failed server and an exception indicating what the failure was and returns a new server for the retry. If a new server is unavailable, RA stub 580 throws an exception.

Fig. 7 is a control flow chart illustrating failover software 508 shown in Figs. 5a-b. Failover method 508 is entered as illustrated by circle 700. A failed provider from the list of current providers of services is removed in logic block 701. A getNextProvider method is then called in order to obtain a service provider. The new service provider is then returned in logic block 703 and failover method 508 exits as illustrated by circle 704.

While Figs. 6-7 illustrate embodiments of a replica handler 506, alternate embodiments include the following functions or combinations thereof implemented in a round robin manner.

First, a list of servers or service providers of a service is maintained. Whenever the list needs to be used and the list has not been recently updated, handler 506 contacts a naming service as described below and obtains an up-to-date list of providers.

Second, if handler 506 is about to select a provider from the list and there is an existing RJVM-level connection to the hosting server over which no messages have been received during the last heartbeat period, handler 506 skips that provider. In an embodiment, a server may later recover since death of peer is determined after several such heartbeat periods. Thus, load balancing on the basis of server load is obtained.

Third, when a provider fails, handler 506 removes the provider from the list. This avoids delays caused by repeated attempts to use non-working service providers.

Fourth, if a service is being invoked from a server that hosts a provider of the service, then that provider is used. This facilitates co-location of providers for chained invokes of services.

Fifth, if a service is being invoked within the scope of a transaction and the server acting as transaction coordinator hosts a provider of the service, then that provider is used. This facilitates co-location of providers within a transaction.

5 The failures that can occur during a method invocation may be classified as being either (1) application-related, or (2) infrastructure-related. RA stub 580 will not retry an operation in the event of an application-related failure, since there can be no expectation that matters will improve. In the event of an infrastructure-related failure,
10 RA stub 580 may or may not be able to safely retry the operation. Some initial non-idempotent operation, such as incrementing the value of a field in a database, might have completed. In an embodiment, RA stub 580 will retry after an infrastructure failure only if either (1) the user has declared that the service methods are
15 idempotent, or (2) the system can determine that processing of the request never started. As an example of the latter, RA stub 580 will retry if, as part of load balancing method, stub 580 switches to a service provider whose host has failed. As another example, a RA stub 580 will retry if it gets a negative acknowledgment to a
20 transactional operation.

 A RMI compiler recognizes a special flag that instructs the compiler to generate an RA stub for an object. An additional flag can be used to specify that the service methods are idempotent. In an embodiment, RA stub 580 will use the replica handler described
25 above and illustrated in Fig 5a. An additional flag may be used to specify a different handler. In addition, at the point a service is deployed, i.e., bound into a clustered naming service as described below, the handler may be overridden.

Fig. 5b illustrates another embodiment of the present invention in which an EJB object 551 is used instead of a stub, as shown in Fig. 5a.

III. Replicated JNDI-compliant naming service

5 As illustrated in Fig. 4, access to service providers in architecture 400 is obtained through a JNDI-compliant naming service, which is replicated across architecture 400 so there is no single point of failure. Accordingly, if a processing device which offers a JNDI-compliant naming service fails, another processing
10 device having a replicated naming service is available. To offer an instance of a service, a server advertises a provider of the service at a particular node in a replicated naming tree. In an embodiment, each server adds a RA stub for the provider to a compatible service pool stored at the node in the server's copy of the naming tree. If
15 the type of a new offer is incompatible with the type of offers in an existing pool, the new offer is made pending and a callback is made through a ConflictHandler interface. After either type of offer is retracted, the other will ultimately be installed everywhere. When a client looks up the service, the client obtains a RA stub that contacts
20 the service pool to refresh the client's list of service providers.

 Fig. 4 illustrates a replicated naming service in architecture 400. In an embodiment, servers 302 and 303 offer an example service provider P1 and P2, respectively, and has a replica of the naming service tree 402 and 403, respectively. The node
25 acme.eng.example in naming service tree 402 and 403 has a service pool 402a and 403a, respectively, containing a reference to Example service provider P1 and P2. Client 304 obtains a RA stub 304e by

doing a naming service lookup at the acme.eng.example node. Stub 304e contacts an instance of a service pool to obtain a current list of references to available service providers. Stub 304e may switch between the instances of a service pool as needed for load-balancing and failover.

Stubs for the initial context of the naming service are replica-aware or Smart stubs which initially load balance among naming service providers and switch in the event of a failure. Each instance of the naming service tree contains a complete list of the current naming service providers. The stub obtains a fresh list from the instance it is currently using. To bootstrap this process, the system uses Domain Naming Service ("DNS") to find a (potentially incomplete) initial list of instances and obtains the complete list from one of them. As an example, a stub for the initial context of the naming service can be obtained as follows:

```
Hashtable env = new Hashtable();
env.put(Context.PROVIDER_URL, "t3://acmeCluster:7001");
env.put(Context.INITIAL_CONTEXT_FACTORY,
"weblogic.jndi.WebLogicInitialContextFactor");
Context ctx = new InitialContext(env);
```

Some subset of the servers in an architecture have been bound into DNS under the name acmeCluster. Moreover, an application is still able to specify the address of an individual server, but the application will then have a single point of failure when the application first attempts to obtain a stub.

A reliable multicast protocol is desirable. In an embodiment, provider stubs are distributed and replicated naming trees are created by an IP multicast or point-to-point protocol. In an IP multicast

embodiment, there are three kinds of messages: Heartbeats, Announcements, and StateDumps. Heartbeats are used to carry information between servers and, by their absence, to identify failed servers. An Announcement contains a set of offers and retractions
5 of services. The Announcements from each server are sequentially numbered. Each receiver processes an Announcement in order to identify lost Announcements. Each server includes in its Heartbeats the sequence number of the last Announcement it has sent. Negative Acknowledgments ("NAKs") for a lost Announcement are
10 included in subsequent outgoing Heartbeats. To process NAKs, each server keeps a list of the last several Announcements that the server has sent. If a NAK arrives for an Announcement that has been deleted, the server sends a StateDump, which contains a complete list of the server's services and the sequence number of its next
15 Announcement. When a new server joins an existing architecture, the new server NAKs for the first message from each other server, which results in StateDumps being sent. If a server does not receive a Heartbeat from another server after a predetermined period of time, the server retracts all services offered by the server not generating a
20 Heartbeat.

IV. Programming Models

Applications used in the architecture illustrated in Figs. 3-5 use one of three basic programming models: (1) stateless or direct, (2) stateless factory or indirect, or (3) stateful or targeted, depending
25 on the way the application state is to be treated. In the stateless

model, a Smart stub returned by a naming-service lookup directly references service providers.

```
Example e = (Example) ctx.lookup("acme.eng.example");  
result1 = e.example(37);  
5      result2 = e.example(38);
```

In this example, the two calls to example may be handled by different service providers since the Smart stub is able to switch between them in the interests of load balancing. Thus, the Example service object cannot internally store information on behalf of the application. Typically the stateless model is used only if the provider is stateless. As an example, a pure stateless provider might compute some mathematical function of its arguments and return the result. Stateless providers may store information on their own behalf, such as for accounting purposes. More importantly, stateless providers may access an underlying persistent storage device and load application state into memory on an as-needed basis. For example, in order for example to return the running sum of all values passed to it as arguments, example might read the previous sum from a database, add in its current argument, write the new value out, and then return it. This stateless service model promotes scalability.

In the stateless factory programming model, the Smart stub returned by the lookup is a factory that creates the desired service providers, which are not themselves Smart stubs.

```
ExampleFactory gf = (ExampleFactory)  
25      ctx.lookup("acme.eng.example");  
      Example e = gf.create();  
      result1 = e.example(37);  
      result2 = e.example(38);
```

In this example, the two calls to example are guaranteed to be handled by the same service provider. The service provider may therefore safely store information on behalf of the application. The stateless factory model should be used when the caller needs to
5 engage in a "conversation" with the provider. For example, the caller and the provider might engage in a back-and-forth negotiation. Replica-aware stubs are generally the same in the stateless and stateless factory models, the only difference is whether the stubs refer to service providers or service provider factories.

10 A provider factory stub may failover at will in its effort to create a provider, since this operation is idempotent. To further increase the availability of an indirect service, application code must contain an explicit retry loop around the service creation and invocation.

```
15     while (true) {  
        try {  
            Example e = gf.create();  
            result1 = e.example(37);  
            result2 = e.example(38);  
20         break;  
        } catch (Exception e) {  
            if (!retryWarranted(e))  
                throw e;  
        }  
25     }
```

This would, for example, handle the failure of a provider e that was successfully created by the factory. In this case, application code should determine whether non-idempotent operations

completed. To further increase availability, application code might attempt to undo such operations and retry.

In the stateful programming model, a service provider is a long-lived, stateful object identified by some unique system-wide key.

5 Examples of "entities" that might be accessed using this model include remote file systems and rows in a database table. A targeted provider may be accessed many times by many clients, unlike the other two models where each provider is used once by one client. Stubs for targeted providers can be obtained either by direct lookup,
10 where the key is simply the naming-service name, or through a factory, where the key includes arguments to the create operation. In either case, the stub will not do load balancing or failover. Retries, if any, must explicitly obtain the stub again.

There are three kinds of beans in EJB, each of which maps to
15 one of the three programming models. Stateless session beans are created on behalf of a particular caller, but maintain no internal state between calls. Stateless session beans map to the stateless model. Stateful session beans are created on behalf of a particular caller and maintain internal state between calls. Stateful session beans map to
20 the stateless factory model. Entity beans are singular, stateful objects identified by a system-wide key. Entity beans map to the stateful model. All three types of beans are created by a factory called an EJB home. In an embodiment, both EJB homes and the beans they create are referenced using RMI. In an architecture as
25 illustrated in Figs. 3-5, stubs for an EJB home are Smart stubs. Stubs for stateless session beans are Smart stubs, while stubs for stateful session beans and entity beans are not. The replica handler to use for an EJB-based service can be specified in its deployment descriptor.

To create an indirect RMI-based service, which is required if the object is to maintain state on behalf of the caller, the application code must explicitly construct the factory. A targeted RMI-based service can be created by running the RMI compiler without any special flags and then binding the resulting service into the replicated naming tree. A stub for the object will be bound directly into each instance of the naming tree and no service pool will be created. This provides a targeted service where the key is the naming-service name. In an embodiment, this is used to create remote file systems.

V. Hardware and Software Components

Fig. 8 shows hardware and software components of an exemplary server and/or client as illustrated in Figs. 3-5. The system of Fig. 8 includes a general-purpose computer 800 connected by one or more communication mediums, such as connection 829, to a LAN 840 and also to a WAN, here illustrated as the Internet 880. Through LAN 840, computer 800 can communicate with other local computers, such as a file server 841. In an embodiment, file server 801 is server 303 as illustrated in Fig. 3. Through the Internet 880, computer 800 can communicate with other computers, both local and remote, such as World Wide Web server 881. In an embodiment, Web server 881 is server 303 as illustrated in Fig. 3. As will be appreciated, the connection from computer 800 to Internet 880 can be made in various ways, e.g., directly via connection 829, or through local-area network 840, or by modem (not shown).

Computer 800 is a personal or office computer that can be, for example, a workstation, personal computer, or other single-user or

multi-user computer system; an exemplary embodiment uses a Sun SPARC-20 workstation (Sun Microsystems, Inc., Mountain View, CA). For purposes of exposition, computer 800 can be conveniently divided into hardware components 801 and software components 5 802; however, persons of ordinary skill in the art will appreciate that this division is conceptual and somewhat arbitrary, and that the line between hardware and software is not a hard and fast one. Further, it will be appreciated that the line between a host computer and its attached peripherals is not a hard and fast one, and that in particular, 10 components that are considered peripherals of some computers are considered integral parts of other computers. Thus, for example, user I/O 820 can include a keyboard, a mouse, and a display monitor, each of which can be considered either a peripheral device or part of the computer itself, and can further include a local printer, which is 15 typically considered to be a peripheral. As another example, persistent storage 808 can include a CD-ROM (compact disc read-only memory) unit, which can be either peripheral or built into the computer.

Hardware components 801 include a processor (CPU) 805, 20 memory 806, persistent storage 808, user I/O 820, and network interface 825 which are coupled to bus 810. These components are well understood by those of skill in the art and, accordingly, need be explained only briefly here.

Processor 805 can be, for example, a microprocessor or a 25 collection of microprocessors configured for multiprocessing.

Memory 806 can include read-only memory (ROM), random-access memory (RAM), virtual memory, or other memory technologies, singly or in combination. Persistent storage 808 can include, for example, a magnetic hard disk, a floppy disk, or other

5 persistent read-write data storage technologies, singly or in combination. It can further include mass or archival storage, such as can be provided by CD-ROM or other large-capacity storage technology. (Note that file server 841 provides additional storage capability that processor 805 can use.)

10 User I/O (input/output) hardware 820 typically includes a visual display monitor such as a CRT or flat-panel display, an alphanumeric keyboard, and a mouse or other pointing device, and optionally can further include a printer, an optical scanner, or other devices for user input and output.

15 Network I/O hardware 825 provides an interface between computer 800 and the outside world. More specifically, network I/O 825 lets processor 805 communicate via connection 829 with other processors and devices through LAN 840 and through the Internet 880.

20 Software components 802 include an operating system 850 and a set of tasks under control of operating system 310, such as a Java™ application program 860 and, importantly, JVM software 354 and kernel 355. Operating system 310 also allows processor 805 to control various devices such as persistent storage 808, user I/O 820, and network interface 825. Processor 805 executes the software of operating system 310, application 860, JVM 354 and kernel 355 in conjunction with memory 806 and other components of computer system 800. In an embodiment, software 802 includes network
25 software 302a, JVM1, RJVM2 and RJVM3, as illustrated in server 302 of Fig. 3c. In an embodiment, Java™ application program 860 is Java™ application 302c as illustrated in Fig. 3c.

Persons of ordinary skill in the art will appreciate that the system of Fig. 8 is intended to be illustrative, not restrictive, and that a wide variety of computational, communications, and information devices can be used in place of or in addition to what is shown in Fig. 8. For example, connections through the Internet 880 generally involve packet switching by intermediate router computers (not shown), and computer 800 is likely to access any number of Web servers, including but by no means limited to computer 800 and Web server 881, during a typical Web client session.

The foregoing description of the preferred embodiments of the present invention has been provided for the purposes of illustration and description. It is not intended to be exhaustive or to limit the invention to the precise forms disclosed. Obviously, many modifications and variations will be apparent to practitioners skilled in the art. The embodiments were chosen and described in order to best explain the principles of the invention and its practical applications, thereby enabling others skilled in the art to understand the invention for various embodiments and with the various modifications as are suited to the particular use contemplated. It is intended that the scope of the invention be defined by the following claims and their equivalents.

CLAIMS

What is claimed is:

1. A distributed processing system, comprising:
a communication medium;
5 a first processing device, coupled to the communication medium, having a first software program emulating a processing device ("JVM1") including a first kernel software layer, having a data structure ("RJVM2"); and,
a second processing device, coupled to the communication
10 medium, having a first software program emulating a processing device ("JVM2") including a first kernel software layer, having a first data structure ("RJVM1"), wherein a message from the first processing device is transferred to the second processing device through the first kernel layer and first software program in the first
15 processing device to the first software program and the kernel software layer in the second processing device.
2. The distributed processing system of claim 1, wherein the first software program in the first processing device is a Java™ virtual machine ("JVM") and the data structure in the first processing
20 device is a remote Java™ virtual machine ("RJVM") and wherein,
the first software program in the second processing device is a Java™ virtual machine ("JVM") and the data structure in the second processing device is a remote Java™ virtual machine ("RJVM") and wherein the RJVM in the second processing device corresponds to
25 the JVM in the first processing device.

3. The distributed processing system of claim 2, wherein the first kernel software layer in the first processing device includes a socket manager software component.

5 4. The distributed processing system of claim 2, wherein the first kernel software layer in the first processing device includes a thread manager software component.

5. The distributed processing system of claim 2, wherein the RJVM in the first processing device includes a message routing software component.

10 6. The distributed processing system of claim 2, wherein the RJVM in the first processing device includes a message compression software component.

15 7. The distributed processing system of claim 2, wherein the RJVM in the first processing device includes a peer-gone detection software component.

8. The distributed processing system of claim 2, wherein the first processing device communicates with the second processing device using a protocol selected from the group consisting of TCP, SSL, HTTP tunneling, and ILOP tunneling.

9. The distributed processing system of claim 1, wherein the first processing device includes a memory for storing a Java™ application.

5 10. The distributed processing system of claim 1, wherein the first processing device is a client and the second processing device is a server for providing a service to the client in response to a client request.

11. The distributed processing system of claim 1, wherein the first processing device is a peer of the second processing device.

10 12. The distributed processing system of claim 10, further comprising:

a second communication medium coupled to the second processing device; and

15 a third processing device, coupled to the second communication medium, having 1) a software program emulating a processing device ("JVM3"), and 2) a first kernel software layer having a first data structure ("RJVM1"), and a second data structure ("RJVM2").

20 13. The distributed processing system of claim 12, wherein the second processing device forwards a message from the first processing device to the third processing device.

14. The distributed processing system of claim 13, wherein the first kernel software layer in the first processing device includes an abbreviation table for abbreviating the message and the third processing device includes a duplicate abbreviation table for reading
5 the message.

15. The distributed processing system of claim 13, wherein the third processing device is coupled to the first communication medium, and wherein the message is transferred to the third processing device through the first kernel software layer in the first
10 processing device to the first kernel software layer in the third processing device.

16. The distributed processing system of claim 1, wherein the first processing device includes a stub.

17. The distributed processing system of claim 16, wherein
15 the stub includes a replica-handler.

18. The distributed processing system of claim 17, wherein the replica-handler includes a load balancing software component and a failover software component.

19. The distributed processing system of claim 1, wherein
20 the first processing device includes an Enterprise Java™ Bean object.

20. The distributed processing system of claim 1, wherein the first processing device includes a naming tree having a pool of RA stubs stored at a node of the tree and the second processing device includes a duplicate of the naming tree.

5 21. The distributed processing system of claim 1, wherein the first processing device includes a multicast program for distributing 1) a RA stub to the first and the second processing device, and 2) a naming tree to the first and the second processing device.

10 22. The distributed processing system of claim 1, wherein the first processing device includes an application program coded in a stateless program model.

23. The distributed processing system of claim 22, wherein the application program includes a stateless session bean.

15 24. The distributed processing system of claim 1, wherein the first processing device includes an application program coded in an stateless factory program model.

25. The distributed processing system of claim 24, wherein the application program includes a stateful session bean.

26. The distributed processing system of claim 1, wherein the first processing device includes an application program coded in a stateful program model.

27. The distributed processing system of claim 26, wherein
5 the application program includes an entity session bean.

28. An article of manufacture, including an information storage medium wherein is stored, comprising:

a first set of digital information for transferring a message from a first remote Java™ virtual machine in a first processing device to a
10 second Java™ virtual machine in a second processing device, wherein the first remote Java™ virtual machine includes a message routing software component.

29. The article of manufacture of claim 28, wherein the first remote Java™ virtual machine further includes a message
15 compression software component.

30. The article of manufacture of claim 28, wherein the first remote Java™ virtual machine further includes a peer-gone detection software component.

31. The article of manufacture of claim 28, wherein the first
20 set of digital information further includes a thread manager software component and a socket manager software component.

32. The article of manufacture of claim 28, further comprising:

a second set of digital information for transferring the message to a communication medium.

5 33. The article of manufacture of claim 28, further comprising:

a second set of digital information for transferring the message to a communication medium; and

10 a third set of digital information, including a remote method invocation.

34. The article of manufacture of claim 28, further comprising:

a second set of digital information for transferring the message to a communication medium; and

15 a third set of digital information including an enterprise Java™ bean.

35. An electronic signal for providing a message from a first computer on a communication medium to a second computer, wherein the electronic signal is embodied in a processor readable memory and includes a first signal section representing a Java™
- 5 virtual machine ("JVM1") and a remote Java™ virtual machine ("RJVM1") for storing in the first computer and a first Java™ virtual machine ("JVM2") and a first remote Java™ virtual machine ("RJVM2") for storing in the second computer, wherein the first
- 10 RJVM1 in the first computer transfers the message to the first RJVM2 in the second computer using a socket.

1/15

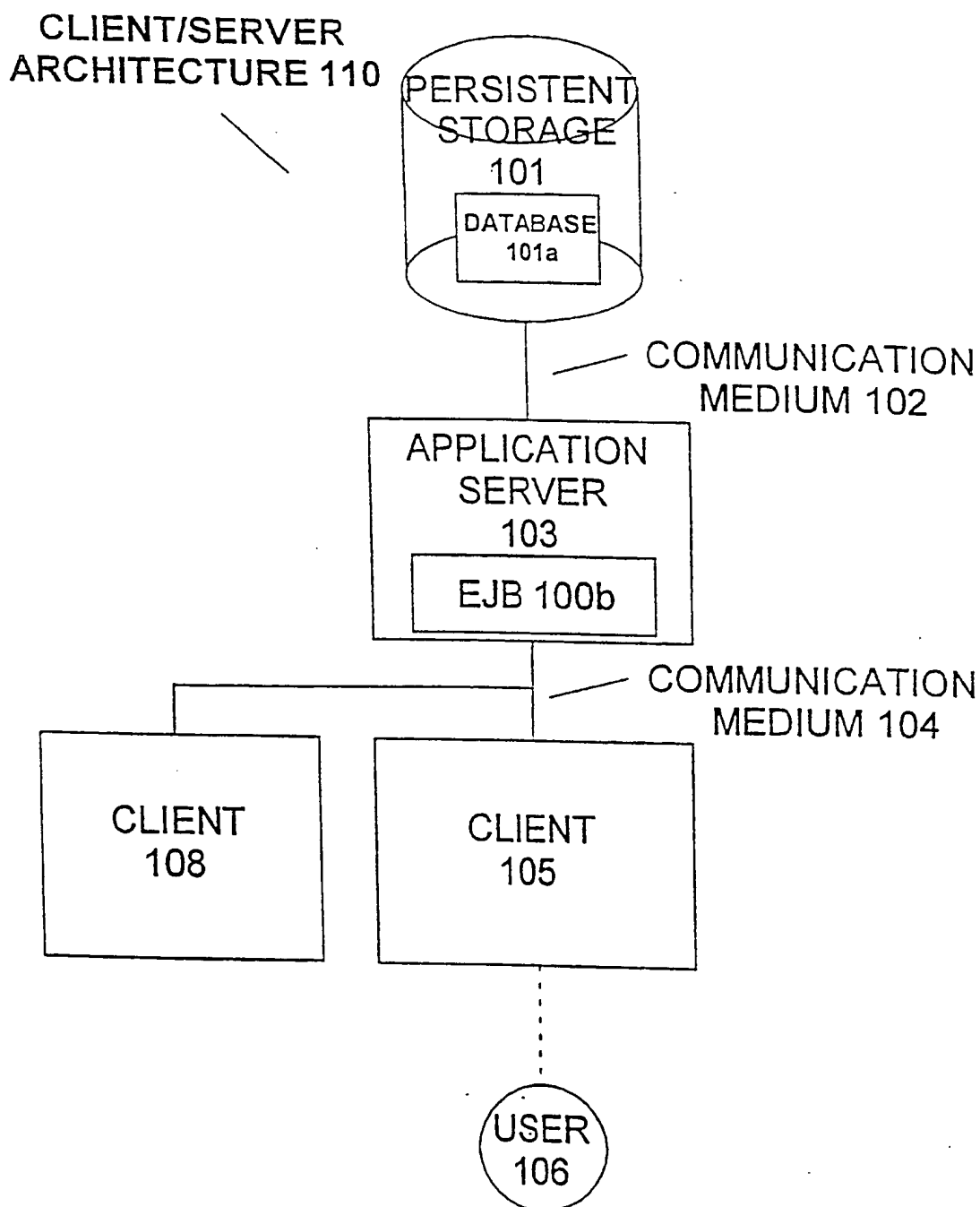


FIG. 1a(Prior Art)

SUBSTITUTE SHEET (RULE 26)

2/15

JAVA ENTERPRISE APIs 100

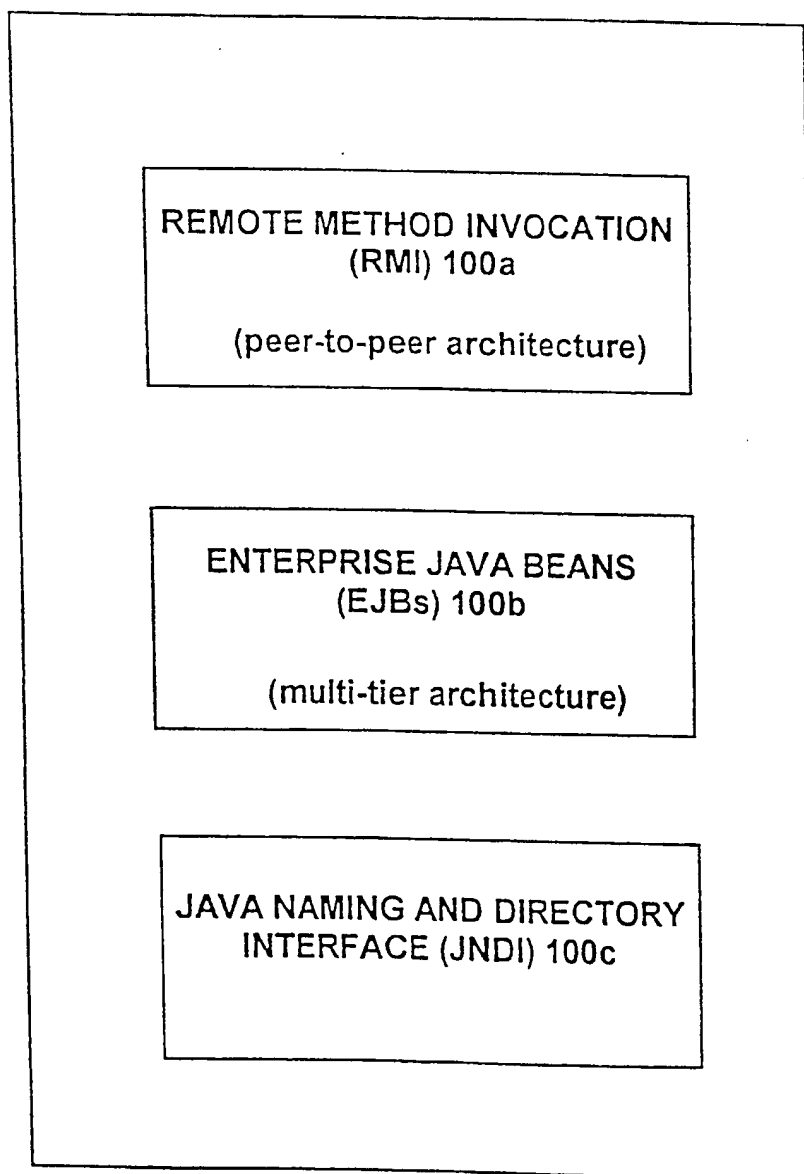


FIG. 1b(Prior Art)

3/15

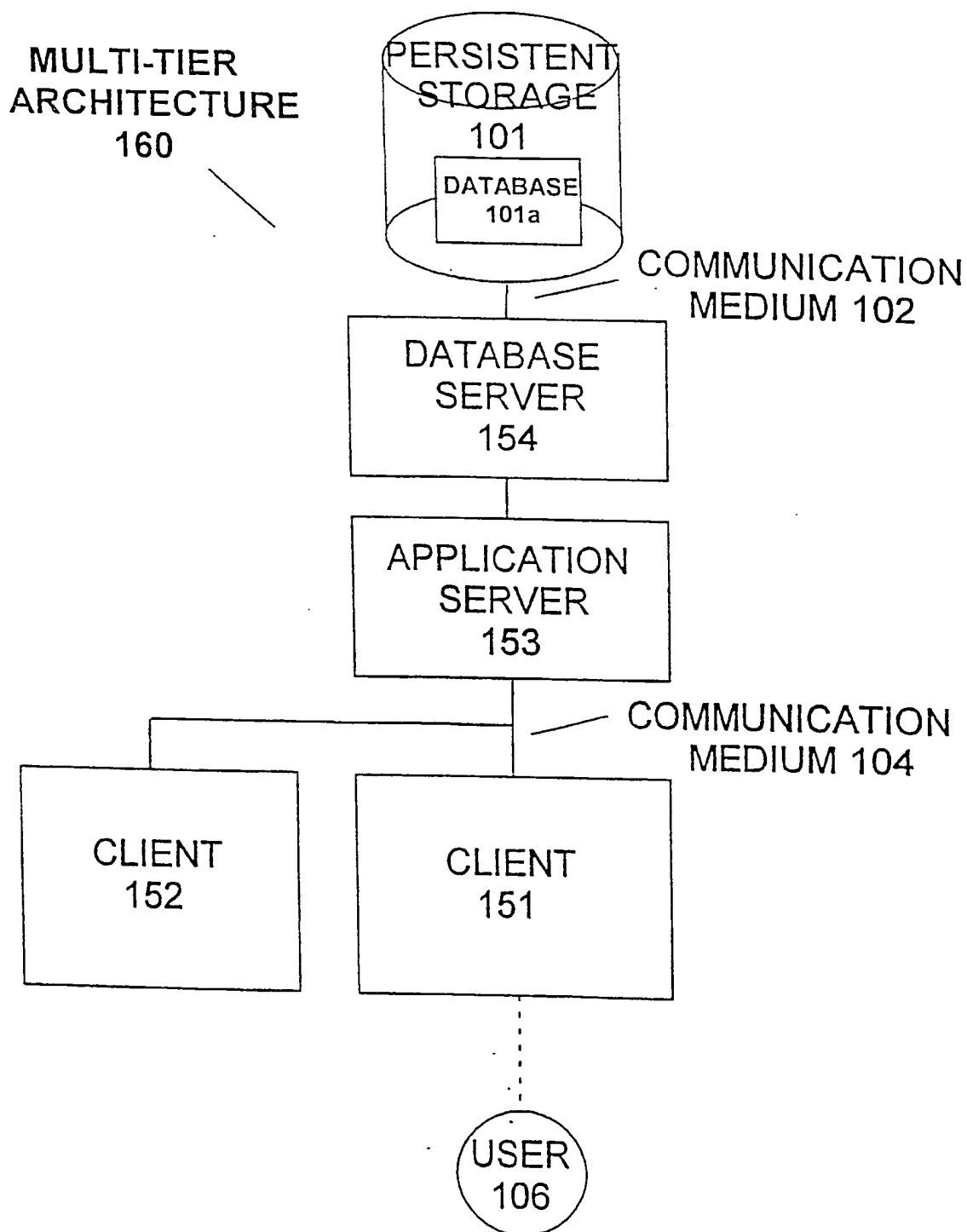


FIG. 1c(Prior Art)

SUBSTITUTE SHEET (RULE 26)

4/15

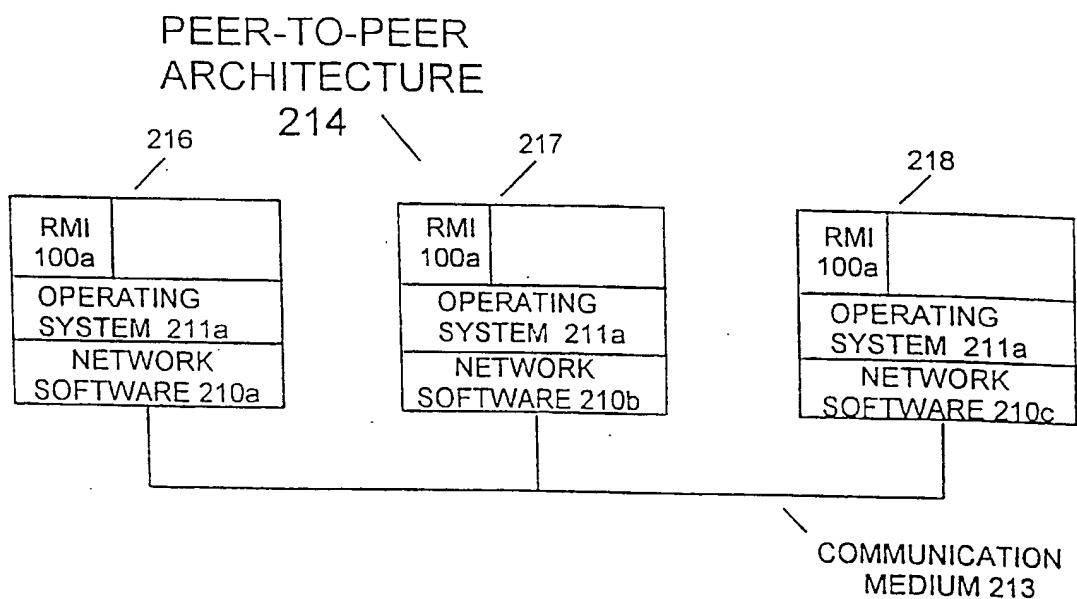


FIG. 2a (Prior Art)

TRANSACTION PROCESSING
(TP) ARCHITECTURE

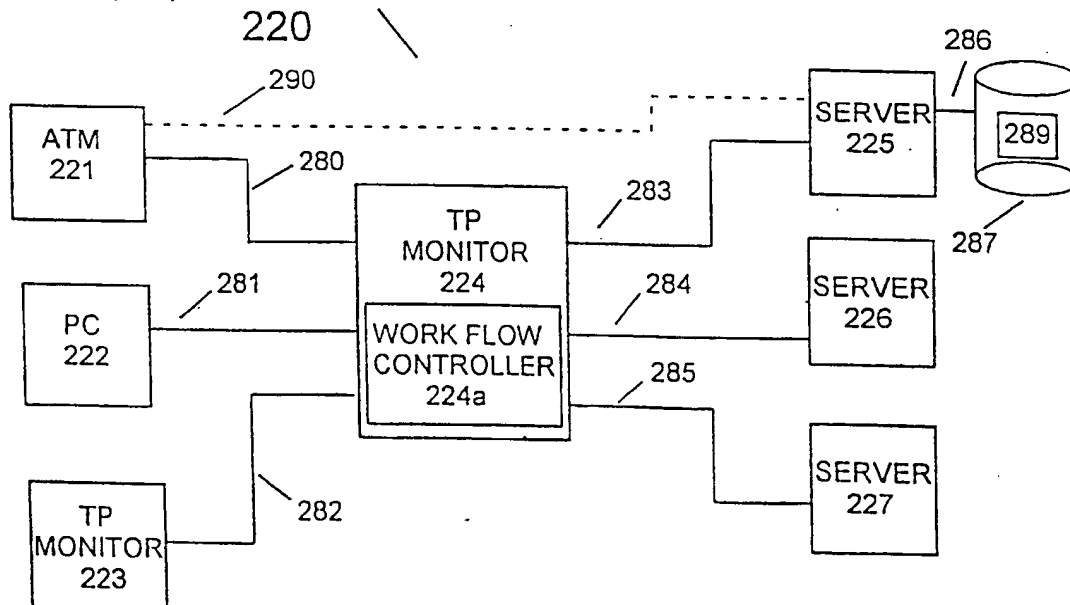


FIG. 2b (Prior Art)

5/15

380

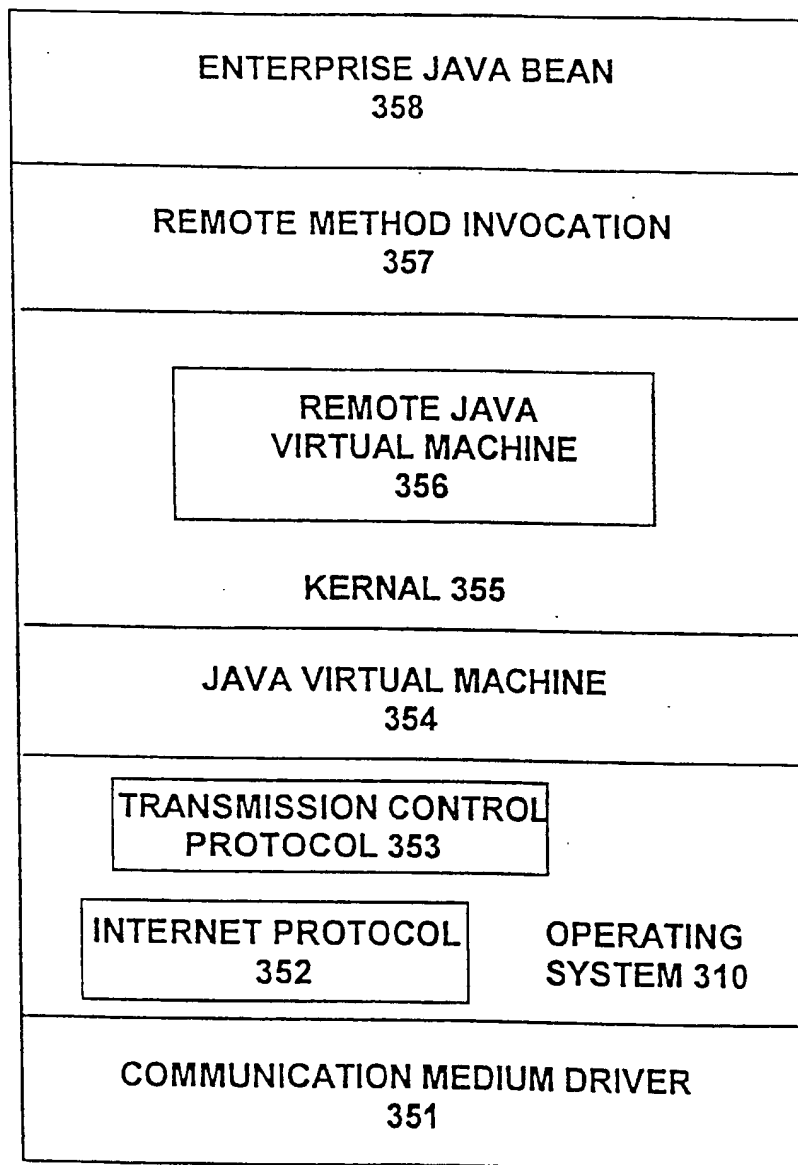


FIG. 3a

6/15

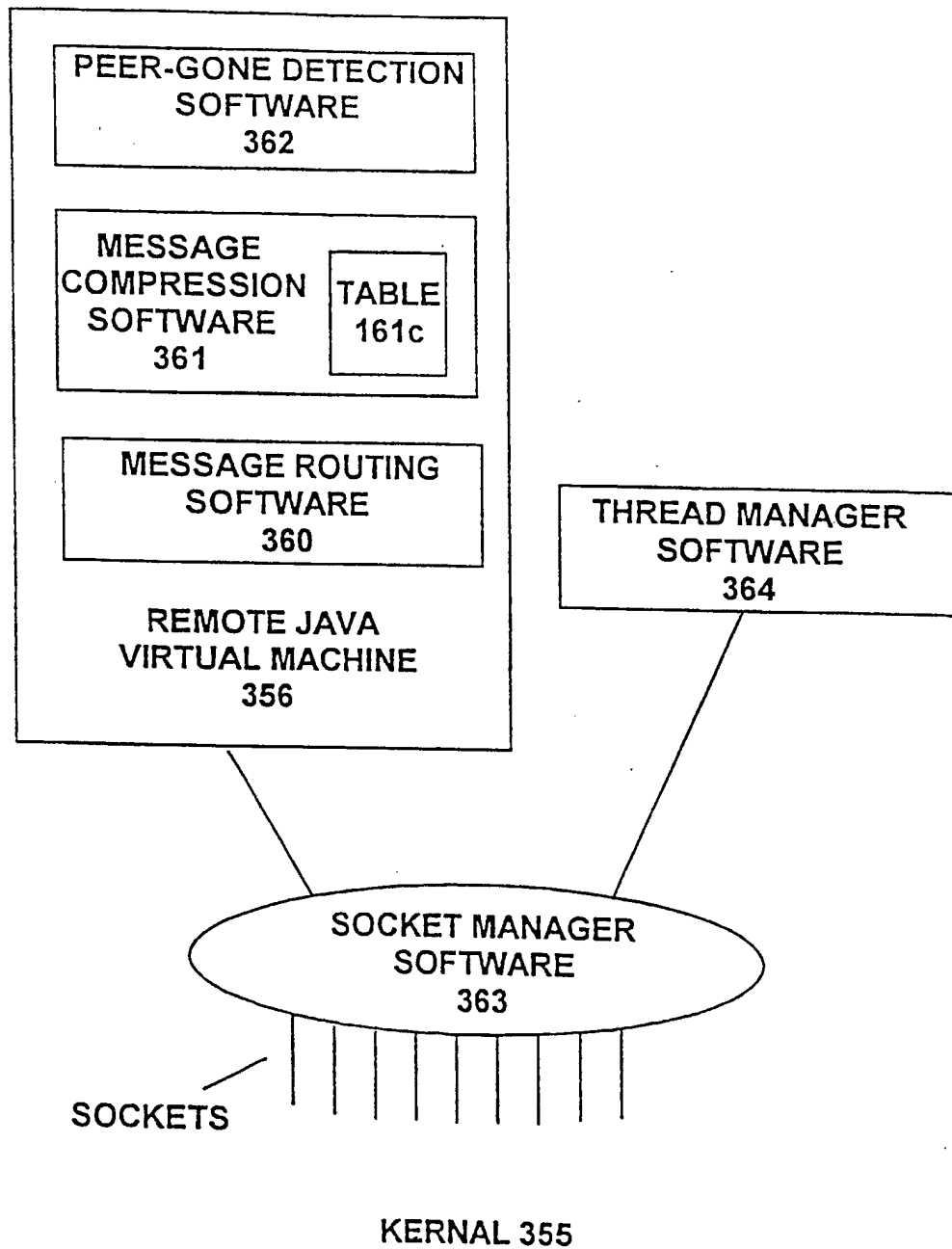


FIG. 3b

SUBSTITUTE SHEET (RULE 26)

7/15

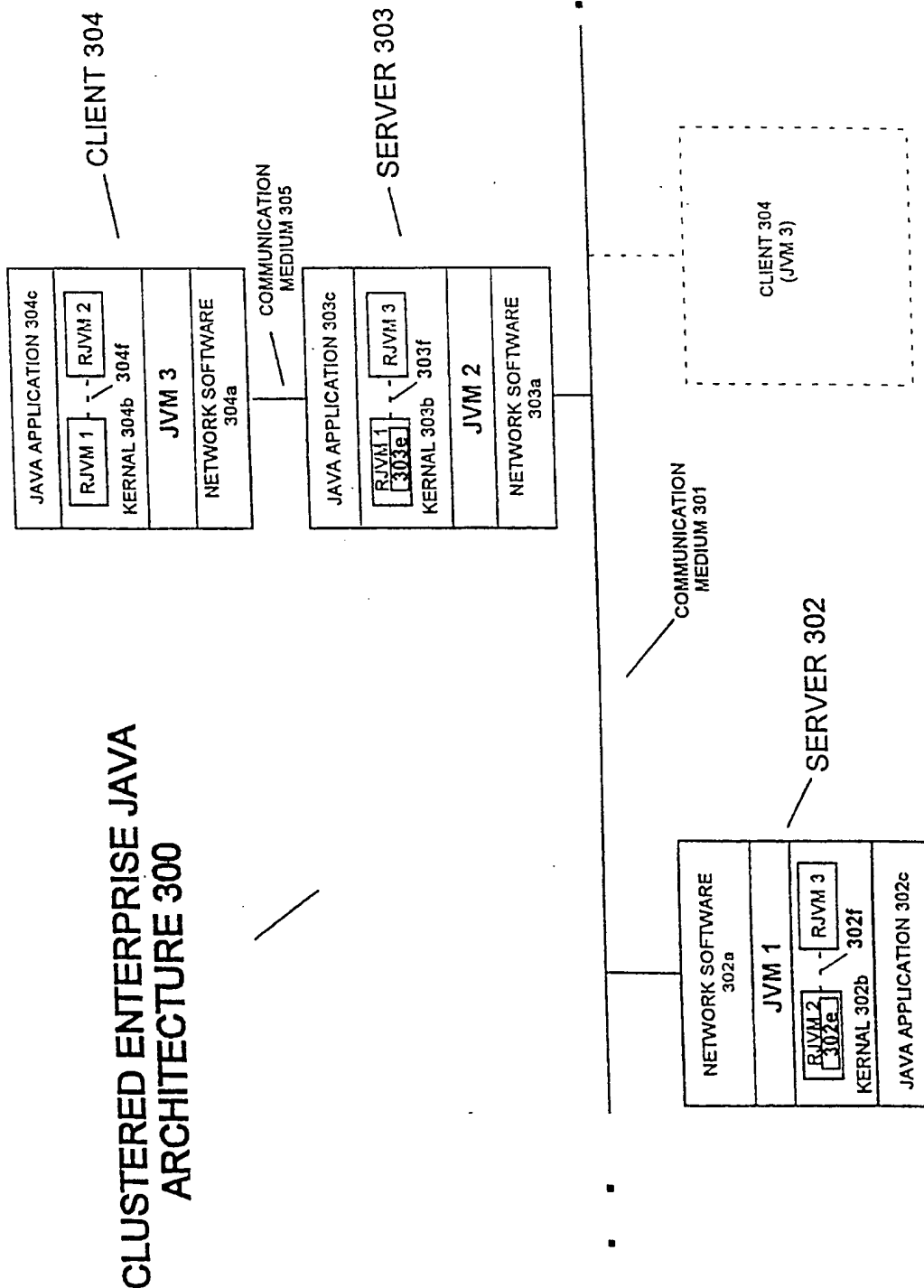


FIG. 3c

CLUSTERED ENTERPRISE JAVA ARCHITECTURE 300

SUBSTITUTE SHEET (RULE 26)

8/15

CLUSTERED ENTERPRISE JAVA ARCHITECTURE NAMING SERVICE 400

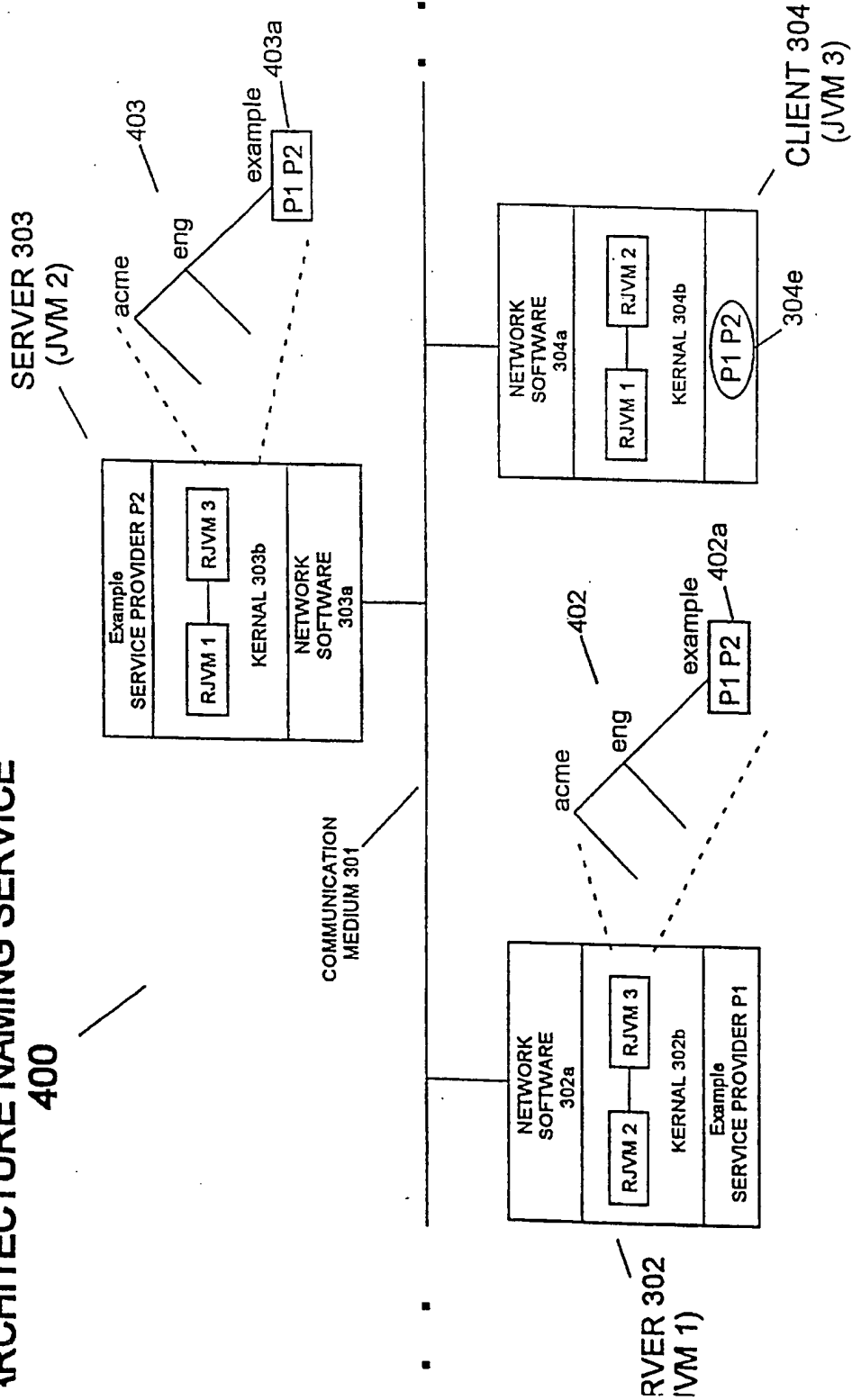


FIG. 4

9/15

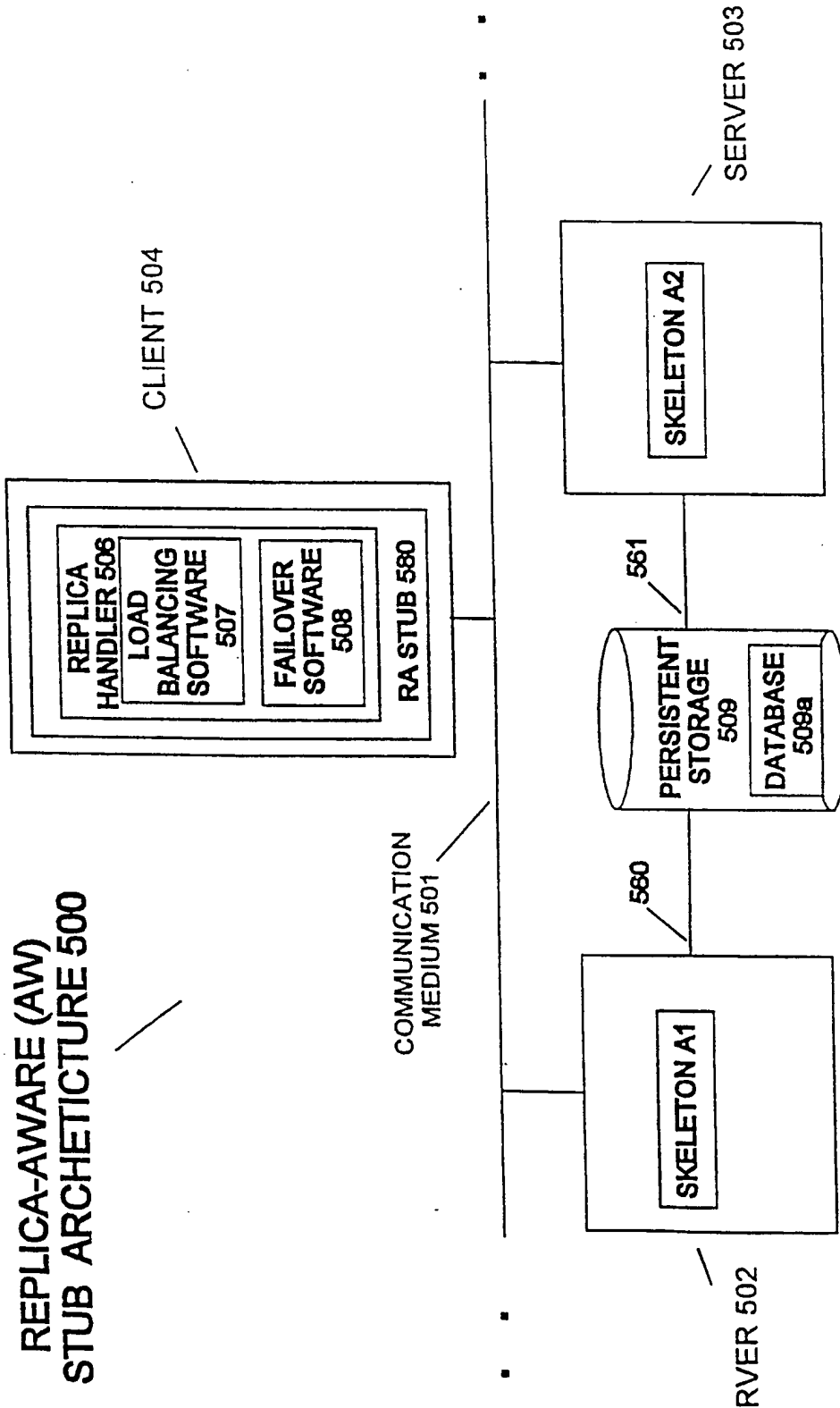


FIG. 5a

10/15

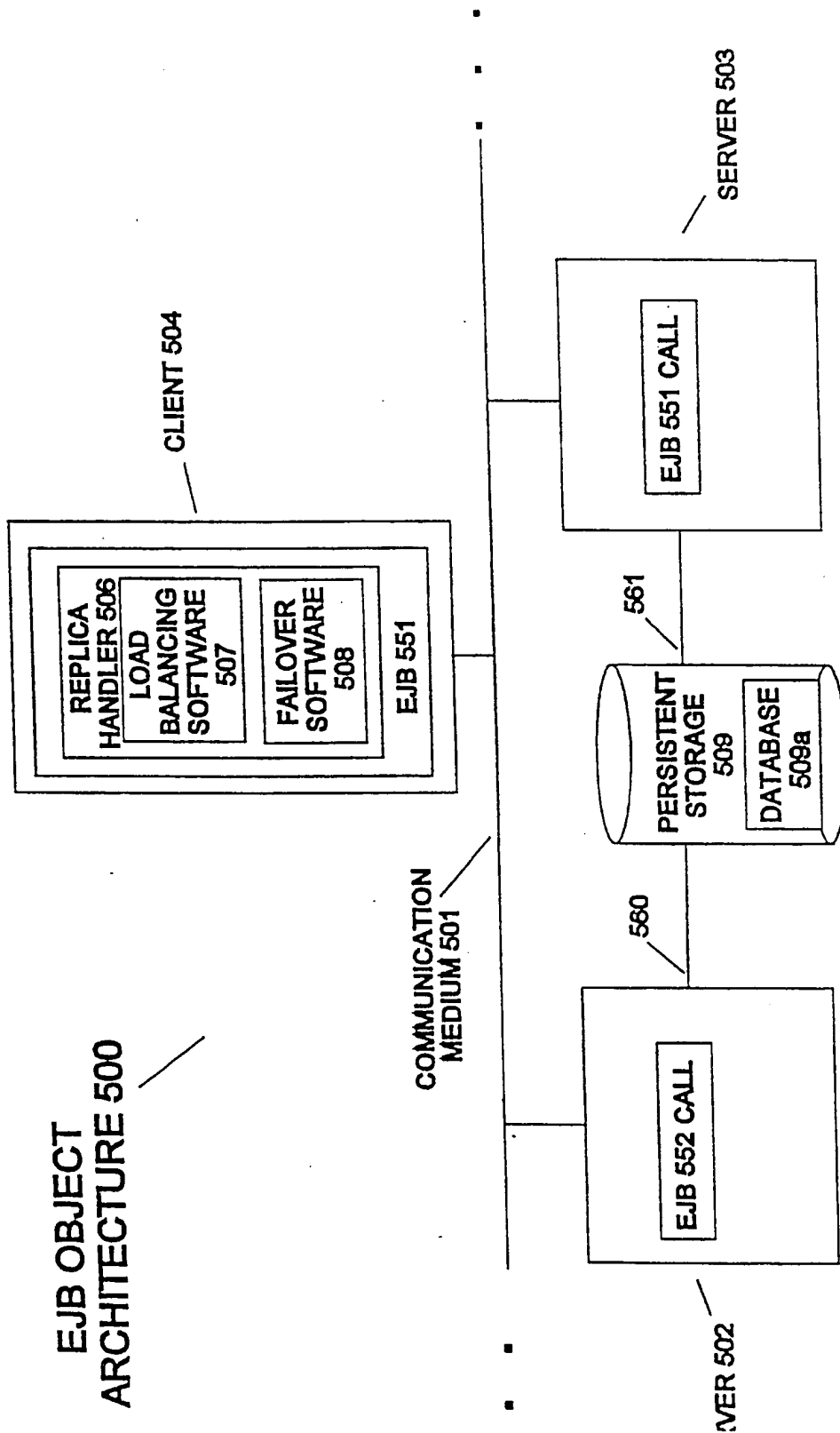


FIG. 5b

11/15

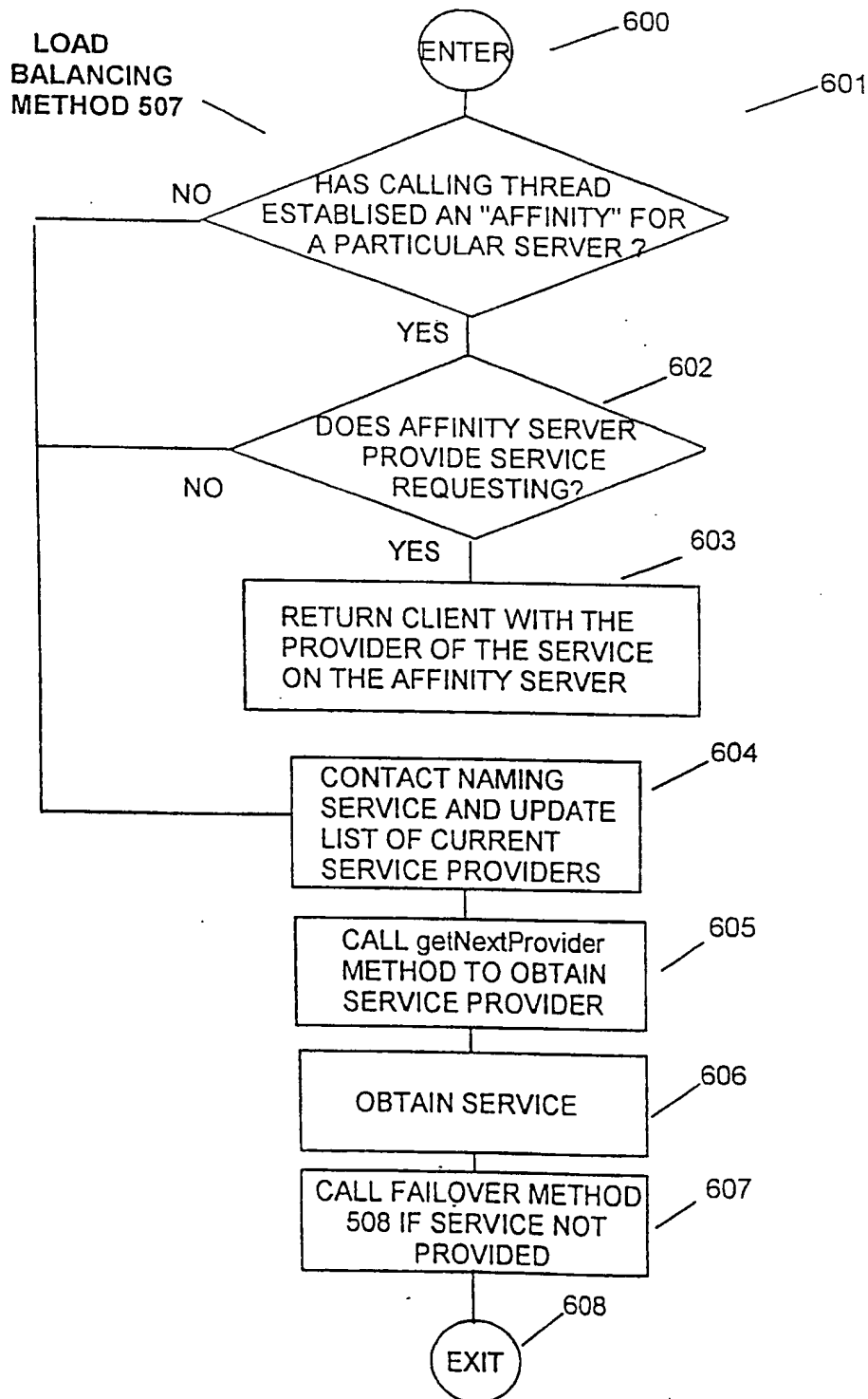
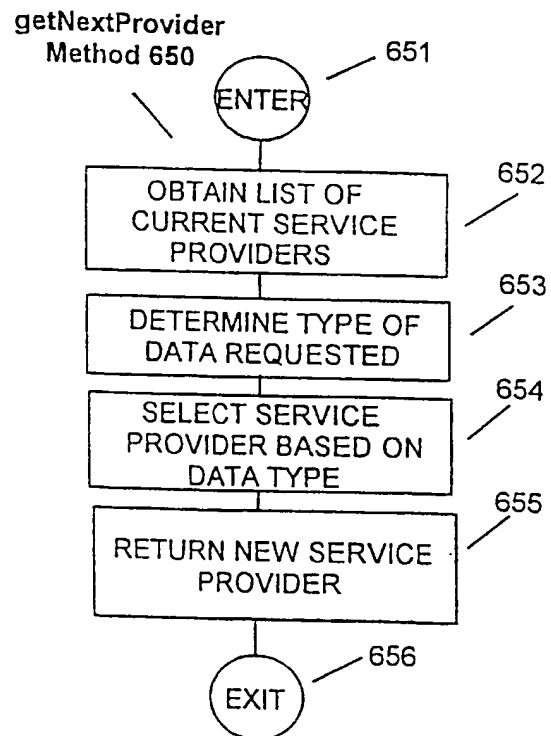
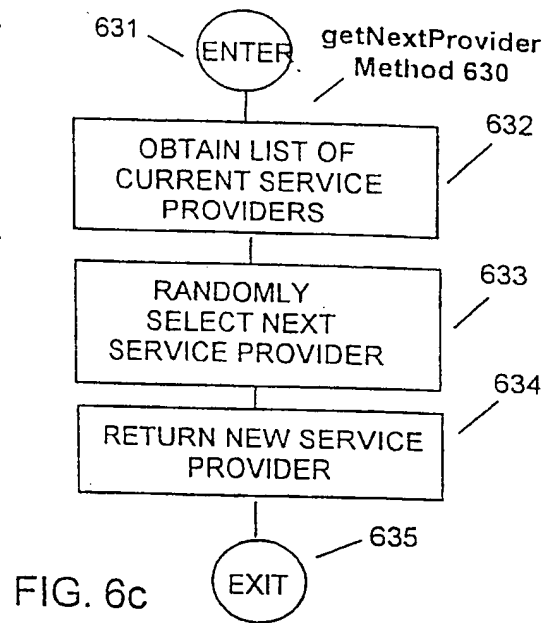
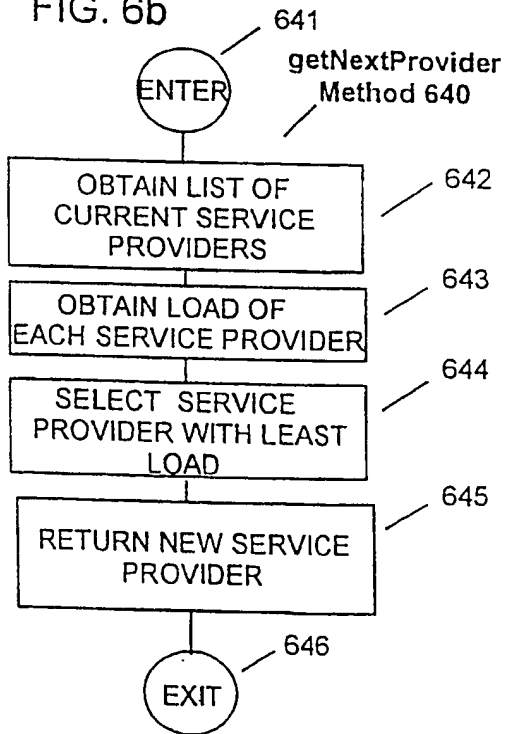
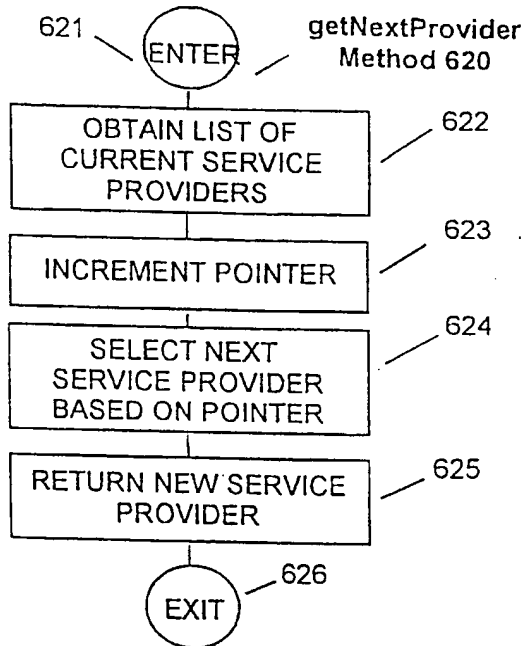


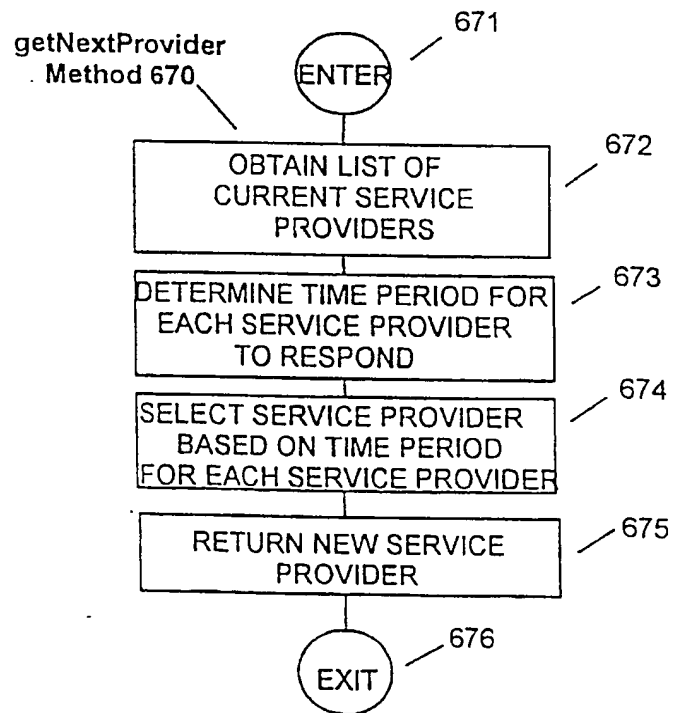
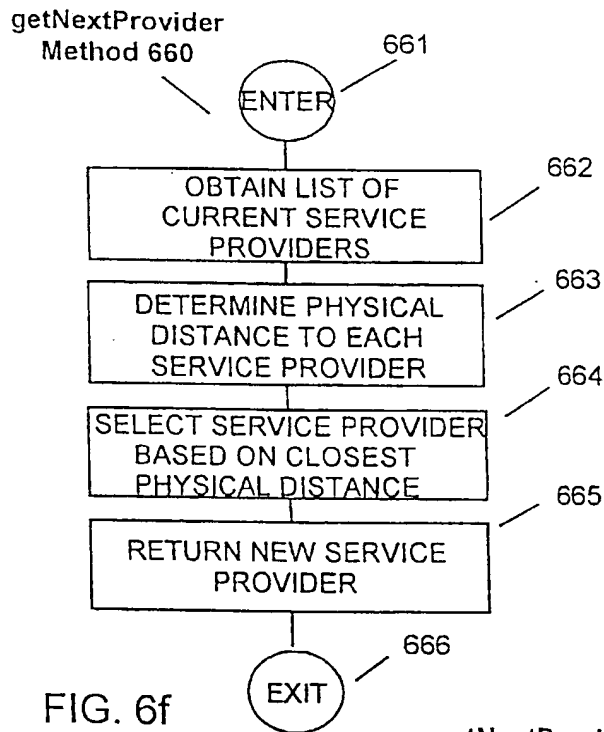
FIG. 6a

SUBSTITUTE SHEET (RULE 26)

12/15



13/15



14/15

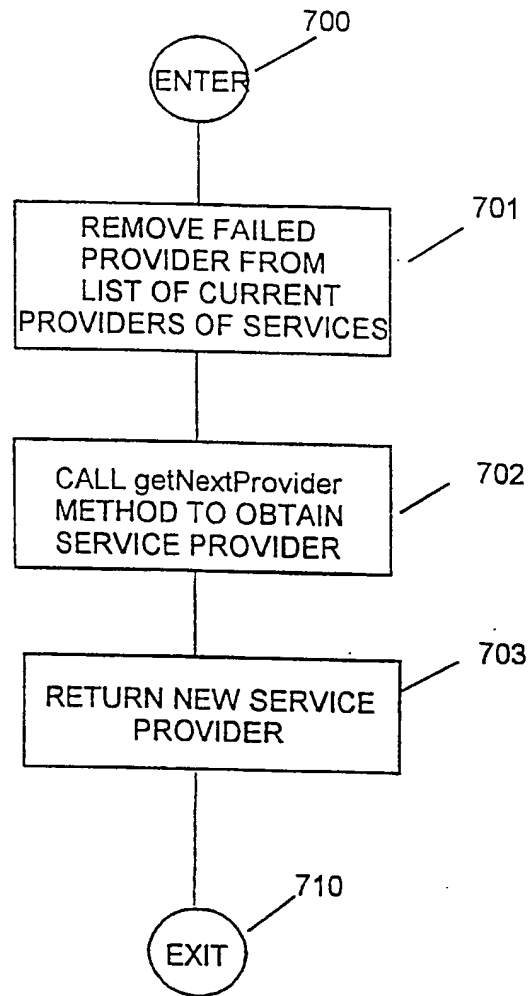
FAILOVER
METHOD 508

FIG. 7

15/15

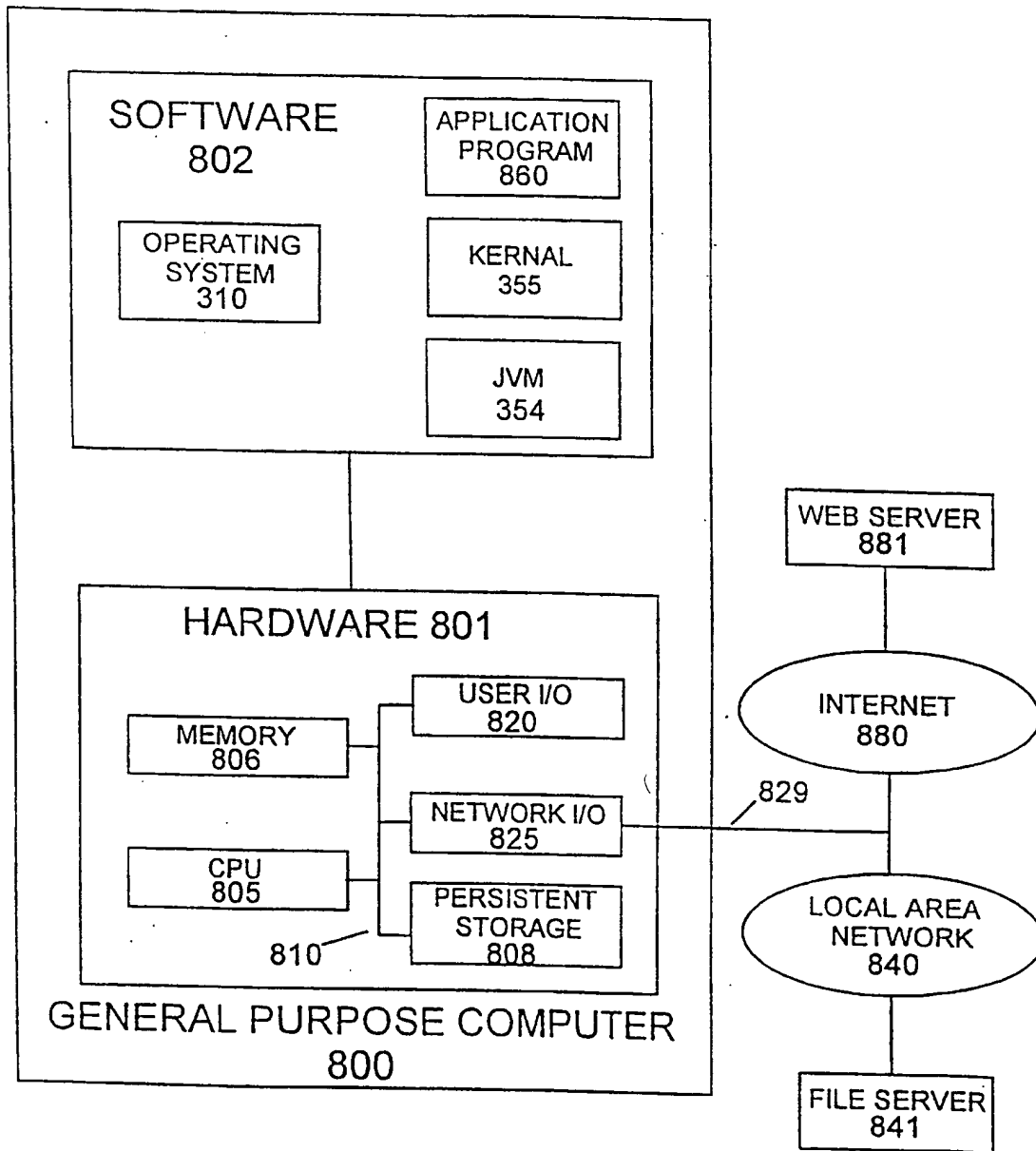


FIG. 8

INTERNATIONAL SEARCH REPORT

International application No.
PCT/US99/24361

A. CLASSIFICATION OF SUBJECT MATTER IPC(6) GO6F 15/16 US CL 709/200 According to International Patent Classification (IPC) or to both national classification and IPC				
B. FIELDS SEARCHED Minimum documentation searched (classification system followed by classification symbols) U.S. 709/200, 201 Documentation searched other than minimum documentation to the extent that such documents are included in the fields searched Electronic data base consulted during the international search (name of data base and, where practicable, search terms used)				
C. DOCUMENTS CONSIDERED TO BE RELEVANT				
Category*	Citation of document, with indication, where appropriate, of the relevant passages	Relevant to claim No.		
X,E	US 6,003,065 A (YAN et al) 14 DECEMBER 1999, see entire document	1-7,9-15,22,26,28-33,35		
<input type="checkbox"/> Further documents are listed in the continuation of Box C. <input type="checkbox"/> See patent family annex.				
<table border="0"> <tr> <td> *A* document defining the general state of the art which is not considered to be of particular relevance *B* earlier document published on or after the international filing date *C* document which may throw doubts on priority claims or which is cited to establish the publication date of another citation or other special reason (as specified) *D* document referring to an oral disclosure, use, exhibition or other means *E* document published prior to the international filing date but later than the priority date claimed </td> <td> *T* later document published after the international filing date or priority date and not in conflict with the application but cited to understand the principle or theory underlying the invention *X* document of particular relevance; the claimed invention cannot be considered novel or cannot be considered to involve an inventive step, when the document is taken alone *Y* document of particular relevance; the claimed invention cannot be considered to involve an inventive step when the document is combined with one or more other such documents, such combination being obvious to a person skilled in the art *Z* document member of the same patent family </td> </tr> </table>			*A* document defining the general state of the art which is not considered to be of particular relevance *B* earlier document published on or after the international filing date *C* document which may throw doubts on priority claims or which is cited to establish the publication date of another citation or other special reason (as specified) *D* document referring to an oral disclosure, use, exhibition or other means *E* document published prior to the international filing date but later than the priority date claimed	*T* later document published after the international filing date or priority date and not in conflict with the application but cited to understand the principle or theory underlying the invention *X* document of particular relevance; the claimed invention cannot be considered novel or cannot be considered to involve an inventive step, when the document is taken alone *Y* document of particular relevance; the claimed invention cannot be considered to involve an inventive step when the document is combined with one or more other such documents, such combination being obvious to a person skilled in the art *Z* document member of the same patent family
A document defining the general state of the art which is not considered to be of particular relevance *B* earlier document published on or after the international filing date *C* document which may throw doubts on priority claims or which is cited to establish the publication date of another citation or other special reason (as specified) *D* document referring to an oral disclosure, use, exhibition or other means *E* document published prior to the international filing date but later than the priority date claimed	*T* later document published after the international filing date or priority date and not in conflict with the application but cited to understand the principle or theory underlying the invention *X* document of particular relevance; the claimed invention cannot be considered novel or cannot be considered to involve an inventive step, when the document is taken alone *Y* document of particular relevance; the claimed invention cannot be considered to involve an inventive step when the document is combined with one or more other such documents, such combination being obvious to a person skilled in the art *Z* document member of the same patent family			
Date of the actual completion of the international search 25 FEBRUARY 2000		Date of mailing of the international search report 21 MAR 2000		
Name and mailing address of the ISA/US Commissioner of Patents and Trademarks Box PCT Washington, D.C. 20231 Facsimile No. (703) 305-3230		Authorized officer FRANK ASTA <i>James R. Matthews</i> Telephone No. (703) 305-3847		

INTERNATIONAL SEARCH REPORT

International Application No

PCT/US 01/47067

A. CLASSIFICATION OF SUBJECT MATTER
IPC 7 H04L29/06

According to International Patent Classification (IPC) or to both national classification and IPC

B. FIELDS SEARCHED

Minimum documentation searched (classification system followed by classification symbols)

IPC 7 H04L

Documentation searched other than minimum documentation to the extent that such documents are included in the fields searched

Electronic data base consulted during the international search (name of data base and, where practical, search terms used)

EPO-Internal, WPI Data, IBM-TDB, INSPEC, COMPENDEX

C. DOCUMENTS CONSIDERED TO BE RELEVANT

Category *	Citation of document, with indication, where appropriate, of the relevant passages	Relevant to claim No.
X	EP 1 081 918 A (HEWLETT PACKARD CO) 7 March 2001 (2001-03-07) page 2, line 40 -page 3, line 17 page 3, line 45 -page 4, line 42 figures 1-5 abstract	10,11, 14-17, 19,20, 23-26
Y		12,13, 18,21, 22,27
Y	EP 0 991 244 A (NORTEL NETWORKS CORP) 5 April 2000 (2000-04-05) column 1, line 45 -column 3, line 2 --- -/--	12,13, 21,22

☒ Further documents are listed in the continuation of box C.

☒ Patent family members are listed in annex.

* Special categories of cited documents:

- *A* document defining the general state of the art which is not considered to be of particular relevance
- *E* earlier document but published on or after the international filing date
- *L* document which may throw doubts on priority claim(s) or which is cited to establish the publication date of another citation or other special reason (as specified)
- *O* document referring to an oral disclosure, use, exhibition or other means
- *P* document published prior to the international filing date but later than the priority date claimed

- *T* later document published after the international filing date or priority date and not in conflict with the application but cited to understand the principle or theory underlying the invention
- *X* document of particular relevance; the claimed invention cannot be considered novel or cannot be considered to involve an inventive step when the document is taken alone
- *Y* document of particular relevance; the claimed invention cannot be considered to involve an inventive step when the document is combined with one or more other such documents, such combination being obvious to a person skilled in the art.
- *&* document member of the same patent family

Date of the actual completion of the international search

1 August 2002

Date of mailing of the international search report

23/08/2002

Name and mailing address of the ISA

European Patent Office, P.B. 5818 Patentlaan 2
NL - 2280 HV Rijswijk
Tel. (+31-70) 340-2040, Tx. 31 651 epo nl.
Fax: (+31-70) 340-3016

Authorized officer

Körbler, G

INTERNATIONAL SEARCH REPORT

International Application No

PCT/US 01/47067

C.(Continuation) DOCUMENTS CONSIDERED TO BE RELEVANT

Category *	Citation of document, with indication, where appropriate, of the relevant passage:	Relevant to claim No.
Y	GRAVESTOCK P.: "Self-Help for Bugs in the Field - A Device-Initiated Upgrade Solution" September 1999 (1999-09), CIRCUIT CELLAR, THE MAGAZINE FOR COMPUTER APPLICATIONS XP002208351 page 2, left-hand column, line 31 -page 3, middle column, line 15 figures 1,2	18,27
A	WO 01 31852 A (BURNETT ALAN MARK ;KEOGH DAVID BRYAN (GB); ROKE MANOR RESEARCH (GB) 3 May 2001 (2001-05-03) page 8, line 20 -page 10, line 18 figure 3	10-27
A	WO 00 67443 A (NILSEN BOERGE ;ERICSSON TELEFON AB L M (SE)) 9 November 2000 (2000-11-09) page 6, line 1 -page 6, line 6	10-27
A	WO 00 28431 A (BEA SYSTEMS INC) 18 May 2000 (2000-05-18) page 16, line 21 -page 17, line 22	10-27

Y = 9, 18, 27

CIRCUIT CELLAR®

THE MAGAZINE FOR COMPUTER APPLICATIONS

XP-002208351

P.D. 00-09-FAA
P. 1-4 (4)

FEATURE ARTICLE

Peter Gravestock

Self-Help for Bugs in the Field

A Device-Initiated Upgrade Solution

Repairing bugs or upgrading firmware after a product is shipped can be quite challenging, especially when the traditional methods (e.g., upgrade disk, FTP download, or visiting engineer) can't be used. With GoAhead's device-initiated architecture, Peter shows us just how easy remote and automatic fixes can be.



In this article, I take you step-by-step through the process of remotely and automatically upgrading the firmware in devices that have been deployed in the field. The solution I outline here can be used by developers of embedded devices and Internet appliances who need to provide field upgrades for their devices.

Developers and manufacturers share the goal of achieving remote programmability while meeting price pressures. Fixing bugs, distributing upgrades efficiently, and creating a vehicle for further sales are all part of that goal. Consider the following situations:

- How do I upgrade my firmware after it has been deployed in the field? I want to perform the upgrade without disturbing the high performance of my product.
- I'd like to add some new product features to my operating system and application software. Some of these systems are installed behind corporate firewalls. Some connect only intermittently to the Internet. Security

is a high priority.

- I want to fix some bugs and we've already shipped. I have deployed thousands of these in devices that are physically hard to access.
- What's the easiest way to get an upgrade out to our customers, when our product doesn't have a disk or CD drive (e.g., cell phones, set-top boxes)?

Has there ever been a hardware or software product that shipped without one bug? Minor, major, catastrophic, or (yes, you guessed it) even Y2K bugs ship with the product.

What kind of solution would enable you to ship your product on time and fix bugs when they arise later? The obvious answer is for the product to upgrade automatically: "Product, heal thyself!"

LIMITED UPGRADE OPTIONS

Options for automatic upgrades so far have been limited and have presented a host of implementation problems. Some manufacturers develop proprietary solutions in-house or use the traditional methods of field upgrading by having customers manually download the upgrade from the web or FTP site, mailing them a disk or CD, sending out an engineer, recalling the product, or doing nothing at all.

There have been some attempts to provide secure upgrades over the Internet in the PC environment. These solutions see the server as the focal point of administrative activity. Clients are added or removed, passwords are changed, and schedules are chosen from the user interface of the server. The server initiates most communication.

History has shown that managing multiple computer systems from the outside is extraordinarily complex. At the level of embedded devices, where the number of discrete systems is in the

millions, these external solutions become completely unwieldy and incapable of providing either the access or capabilities needed to lower support costs or simplify management. An alternative and contrasting solution is to build intelligence into the device enabling the device to manage itself from the inside.

Until now, there has not been an off-the-shelf, device-initiated solution for remotely upgrading operating systems, software, and firmware in embedded systems over the Internet. Recent advances in the embedded market have begun to address this issue.

An ideal upgrade solution would make your embedded devices smarter and your products more reliable, manageable, easier to use, and less expensive to support. An efficient solution would support network devices by automatically ensuring that they have the latest version of software. It would make it easy to securely integrate, deploy, and publish upgrades over the Internet, Intranets, virtual private networks, and dial-up connections.

DEVICE-INITIATED ARCHITECTURE

Making the upgrade process device-initiated is the key to the architecture of the solution outlined in this article. This solution solves some particularly difficult issues, such as firewall penetration, scalability, simplicity, transient connections, efficiency, and so on. Unlike traditional management schemes in which the central console is responsible for tracking all remote devices and initiating contact, in this design the device is autonomous and is responsible for initiating and performing its own upgrades.

A device that can be addressed from an open network is exposed to unauthorized upgrades. With the device-initiated approach, the address of the upgrade server is preprogrammed into the device. The device will only send upgrade requests to the designated server and will only accept upgrades from that

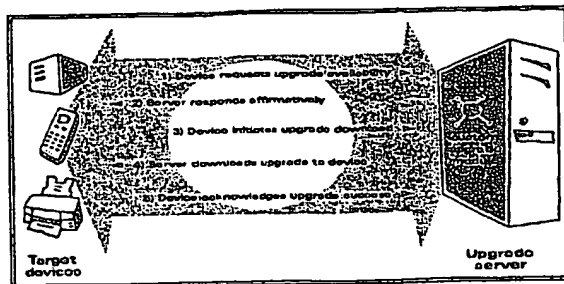


Figure 1—Here's a quick look at the steps of the device-initiated upgrade process.

server.

Typically, servers cannot initiate communication to devices on a network inside a firewall. However, the device-initiated approach presented here ensures that upgrade requests can communicate with the server through firewalls. It uses standard HTTP requests, and most firewalls are configured to allow HTTP requests to the outside world from within the firewall-protected network. With the device-initiated approach, the device sends these requests through the firewall to the server, thus opening the two-way communication channel that is then used for the upgrade transfer.

This article focuses on the device-initiated approach used in an application called GoAhead FieldUpgrader (V.2.0). The application consists of two parts: a development environment for creating device-specific agent software to be embedded in the target device, and a server that publishes upgrades and fulfills device requests.

The agent software is created by the developer using a web-based development environment that runs on Windows NT (and soon on Solaris). The upgrade agent sits between the RTOS and the application to facilitate and speed the integration process. The developer defines one or more components for any device. Each component

contains its server URL, name, version, and polling frequency.

Once developed, this agent on the device polls the upgrade server, which can reside at the location of the network operator, the manufacturer, or any secure site. The agent queries the server for upgrades at predefined intervals. When an upgrade is available, the device initiates the download of the upgrade information and applies it as specified by the manufacturer. The automatic process is illustrated in five steps in Figure 1.

The agent polls the server at regular, predetermined intervals to query for upgrades. There is an embedded JavaScript extension available so the manufacturer can change the interval. (The embedded JavaScript used in this particular solution is a strict subset of JavaScript developed specifically for use in memory-constrained embedded systems.)

The server receives the request and validates the request with the upgrade policies that the manufacturer has defined. These policies control which remote agents should receive upgrades. The policies are based on criteria that may include host names, domains, schedules, current revision of the requesting device, or desired pace of the rollout. A policy can also check the server activity and request that the agent try later so the server doesn't get bogged down with more requests than the network can handle. Manufacturers can use these policies in conjunction with their own support policy to provide upgrades only to beta customers or only to those customers who have purchased support maintenance agreements. The server then checks for upgrades for that device. If the request is validated, the

server notifies the device that a valid upgrade is available.

The device then initiates a request for the upgrade. The server points to the top directory in the tree containing upgrade files, then recurses and archives all the files to create a single file called the upgrade payload. If the upgrade payload is too large,

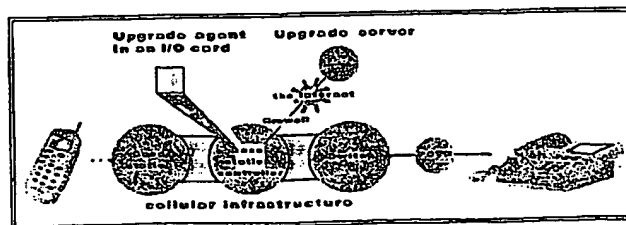


Figure 2—The device-initiated upgrade process can be deployed through a wireless network.

the server divides it into "chunks" so the upgrade can be easily restarted if it is interrupted before completion. If, for example, the server goes down and only three chunks of the upgrade payload have been delivered, the upgrade will continue with the fourth chunk. It will not have to repeat the previous three chunks. The agent then downloads the upgrade payload from the server to the device.

The device authenticates the source of the upgrade using the Diffie-Hellman security keys, reconstructs the payload from the chunks, calculates a 32-bit CRC checksum for each chunk, and performs a higher-level data integrity check using a secure hash algorithm. If the device detects an error in any chunk, it discards that chunk and refetches from the upgrade server. Once all of the chunks have been downloaded and the upgrade has completed successfully, the device notifies the server of the upgrade status.

To give you a better picture of how the pieces fit together in a real environment, Figure 2 depicts where the upgrade server and upgrade agent are deployed in a wireless network.

SECURE UPGRADES OVER THE INTERNET

This solution uses Diffie-Hellman public key exchange to provide encryption keys. The target device and the server each have a public and private key. The device provides its public key to the server, where it is combined with the server's private key to generate a "secret number" that is used to encrypt the upgrade.

Once the target device receives the payload, it decrypts the manifest file. If the decryption is successful, the target device knows that the payload was received from the authorized server. It

then calculates the message digest of the payload and compares it with the value that was sent with the payload. If they match, the target device knows that the payload was not modified en route. If the message digest in the payload doesn't match the number calculated at the target device, the payload is assumed to be suspect and is discarded.

Using this solution, you will no longer need to perform upgrades using FTP to manually retrieve new software, nor send out technicians to visit every remote site to manually perform upgrades.

SAMPLE UPGRADE

Let's walk step-by-step through a sample upgrade of a device. The entire process of creating and running an end-to-end upgrade solution will typically take less than a day.

First, decide what you want to upgrade in the device. For firmware, you can use the upgrade solution in conjunction with your device-specific upgrade utility. If it's the operating system, applications, or data files, the application provides JavaScript with extensions so you can write upgrade scripts that are device-dependent. In Listing 1, you load a DLL, change your upgrade server URL, run the function *doMyUpgrade*, unload your DLL, then reboot the system.

Custom extensions to JavaScript are provided to perform special upgrade tasks. The extensions used in Listing 1 consist of: *loadModule*, *changeServerUrl*, *runFunction*, *unloadModule*, and *reboot*.

To do this task, you need a computer (Pentium class or better) running Windows NT/98 for creation of the agent software using the development environment. You also need a '486 or bet-

ter, running Windows NT to act as the server. In addition, you need:

- RTOS or OS (e.g., VxWorks, Lynx, Windows 95/98/NT)
- TCP/IP stack
- the device in which you want to build upgrade capacity
- GoAhead FieldUpgrader Product Evaluation OR GoAhead UpgradeServer and FieldUpgrader (FieldUpgrader includes the GoAhead DeviceStudio development environment.)

INCORPORATING THE AGENT SOFTWARE

Install and run the development environment on a Windows NT system to create a device-based upgrade agent. Define one or more components for the device. Each component should contain its URL, name, version, and polling frequency. To create the agent:

- use the wizard-based user interface in the development environment to configure the location of your upgrade server (typically an IP address)
- configure the polling frequency. This can be any regular period of time, for example, once a day if you have a small customer base. If you have a large customer base with over 1,000,000 devices, having the devices check for upgrades once a week will spread the network load and allow a single upgrade server to manage the entire device population.
- configure the initial version information for firmware in the device
- select JavaScript if you will be using it
- select the security level you want (you won't need high security if you are working within your own network behind a firewall). Running without security and JavaScript will reduce the memory footprint.

The final step to create the agent is to generate an OS package for your upgrade agent. In Windows NT, for example, this packaging process generates an executable file that is run as an application on the device. In VxWorks, this packaging process generates object files that are linked and compiled within the Tornado development envi-

Listing 1—This short listing is an example of some of the custom Javascript extensions.

```
loadModule("myUpgrade.dll", "myDllOpen");    loads a module(DLL)
changeServerUrl("http://new.server.com");      changes server;
                                              upgrader queries for
                                              upgrades
runFunction("doMyUpgrade");                    runs in-memory function
unloadModule("myUpgrade.dll");                 unloads a module(DLL)
reboot();                                       reboots the system
```

Summary	Breakdown of the upgrade statistics
Activity	Activity per upgrade module
Request log	Log file of each device request
Upgrade log	Log file of each device upgrade
Modules	List of the available upgrade modules
Agents	Lists the IP addresses of requesting devices
Policy denials	List of upgrade denials due to policy

Table 1—You can monitor the upgrade process at the server and view reports.

ronment to generate your VxWorks software image. The packaging step supplies the necessary Diffie-Hellman encryption keys to ensure a secure upgrade.

PUBLISH YOUR UPGRADE

Now that you have completed the agent piece, turn your attention to the upgrade server. The upgrade server runs on a Windows NT system and is deployed in similar fashion to a web server. To publish an upgrade, the manufacturer places a copy of the upgrade file(s) and Diffie-Hellman public keys in a directory that is accessible from the system hosting the upgrade server. Then, from within the upgrade server, select *Create Module* and type in the location of the upgrade file(s) and Diffie-Hellman public keys. The upgrade module contains the upgrade image, security keys, and policy information. Modules support many upgrades for a given component.

Next, supply the version number(s) of the component you want to upgrade and the new version number(s) of the upgraded component. Set the upgrade policy, which controls who is authorized to receive upgrades and how fast you want to roll out upgrades.

Then you're done. The rest is automatic. Your devices that contain the upgrade agent will poll the upgrade server for upgrades, and the server will validate the request against upgrade availability and policy information. The agent will initiate the download from the server. The upgrade will be applied to the target device. The agent will notify the server that the upgrade has completed successfully (see Table 1).

FEATURES AND BENEFITS

This upgrade solution addresses the paramount concerns for upgrading embedded devices and Internet appliances in the field. It is:

- automatic—the end user will need to do nothing
- secure and fast—no hackers will be able to mess with it
- scalable—it will support millions of devices, yet is customizable to the needs of individual devices
- extensible—it enables you to use JavaScript to add policies and send extra agent data back to the server

Some benefits of this solution are that it:

- uses standard HTTP and TCP/IP protocols
- restarts an upgrade if it is interrupted before it is completed
- enables the publisher of the upgrade to have full, centralized control

Manufacturers can offer everything from small bug fixes and reliability patches to new features and huge daily database changes. They can set policies for individual devices to allow policy-based upgrades—for example, to upgrade only those devices whose owners have purchased a maintenance agreement.

The solution is efficient has a small memory footprint, and uses a tiny portion of the CPU. Its simplicity enables you to publish upgrades from small bug fixes to huge daily database changes. Finally, it is flexible, working across multiple platforms and embedded devices.

MAKE THE CONNECTION

Your support load can be reduced if you can effortlessly keep your customers on the very latest software (which also reduces risk as any bugs that do appear can be fixed quickly and preemptively). With the type of solution described above, you can easily integrate, deploy, and publish upgrades over the Internet, Intranets, virtual private networks, and dial-up connections.

A device-initiated architecture provides many advantages over the traditional management model, including

security and flexibility that is not possible through a server-initiated architecture. This type of architecture provides autonomous and mobile clients with the ability to tunnel through corporate firewalls, support for nontethered devices, and the scalability provided by extant web technologies.

Modeled on the world-wide web, with passive servers waiting for connections, and active clients (browsers) making requests, this efficient solution will scale to more devices than you can manufacture and sell.

Peter Gravestock, cofounder and chief technology officer of development at GoAhead Software, is the primary architect for its embedded-management products. Peter's 15 years of development experience encompass many major operating systems. As Technical Lead at PAXUS, he completed the first port of Unix to proprietary Intel 80286 hardware and the IBM AT. You may reach him at peter@goahead.com.

SOURCES

GoAhead FieldUpgrader V.2.0
GoAhead Software Inc.
(425) 453-1900
Fax: (425) 637-1117
www.goahead.com

Circuit Cellar, the Magazine for Computer Applications. Reprinted by permission. For subscription information, call (860) 875-2199 or subscribe@circuitcellar.com.

X. RELATED PROCEEDINGS

None.

Mailing Label
Label 11-B September 2002

Post Office to Addressee

DELIVERY (P) AL USE ONLY

UNITED STATES POSTAL SERVICE®



ER 163796141 US

The following has been received by the USPTO on the date stamped herein.

RECEIVED:

Express Mail PO to Addressee: **ER 163796141 US**

Date Mailed: **11/29/2006**

Docket No. **005168.P002**

Application No. **09/912,636**

CONTENTS:

Transmittal Form sb21 - 1 pg

Fee Transmittal SB/17 - 1 pg

PTO-2038 Payment of Extension Fee on CC ... 7133 \$250.00

APPEAL BRIEF - 974 PAGES

This return receipt postcard - 1 postcard

Certificate of Express Mail PO to Addressee with a copy of this
return receipt postcard - 1 pg

**Express Mail Post Office to Addressee
CERTIFICATE OF MAILING**

Express Mail Post Office to Addressee mailing label number: **EQ 163796141 US**
Date of Deposit: **29 November 2006**

I hereby certify that I am causing this paper or fee, and the documents indicated on the postcard above, to be deposited with the United States Postal Service for "Express Mail Post Office to Addressee" service on the date indicated above and that this paper or fee has been addressed with sufficient postage to:

Mail Stop: Appeal Brief Patents
Commissioner for Patents
P.O. Box 1450
Alexandria, VA 22313-1450

Stephanie W. Roberts

(Typed or printed name of person mailing paper or fee)

(Signature of person mailing paper or fee)

29 November 2006

(Date)